uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Divine Muhivuwomunda
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S.
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Data De-Duplication Through Active Learning
TITRE DE LA THÈSE / TITLE OF THESIS

Herna Vikter
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

Michael Weiss

Liam Peyton

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Data De-Duplication through Active Learning

by

Divine Muhivuwomunda

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the MCS degree in
Computer Science

School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 978-0-494-74192-4
Our file Notre référence
ISBN: 978-0-494-74192-4

Abstract

Data de-duplication concerns the identification and eventual elimination of records, in a particular dataset, that refer to the same entity without necessarily having the same attribute values, nor the same identifying values. Machine Learning techniques have been used to handle data de-duplication. Active Learning using ensemble learning methods is one such technique. An ensemble learning algorithm is used to create, from the same training set, a set of models that are different. Active Learning then iteratively passes unlabeled pairs of records to the created models for labeling as duplicates, or non-duplicates, and selectively picks the pairs that cause most disagreement among the models. The selected pairs of instances are considered to bring most information gain to the learning process. Active Learning thus continuously teaches a learner to find duplicate instances by providing the learner with a better training set.

This thesis evaluates how Active Learning undertakes the task of data de-duplication when Query by Bagging and Query by Boosting algorithms are used. During the evaluation, we investigate the performance of Active Learning in various situations. We study the impact of varying the data size as well as the impact of using different blocking methods, which are methods used to reduce the number of potential duplicates for comparison. We also consider the performance of Active Learning when a synthetic dataset is used versus a real-world dataset.

The experimental results show that Active Learning using Query by Bagging performs well on synthetic datasets and only requires a few iterations to generate a good de-duplication function. The size of the dataset does not seem to have much effect on the results. When the experiment is conducted on real-world data, Active Learning using Query by Bagging still performs well, except when the dataset has a significant amount of noise. However, the learning process for real world data is not as smooth compared to when the synthetic data is used. The performance using Canopy Clustering and Bigram Indexing blocking methods were evaluated and the results show better results for the Bigram Indexing.

Active Learning using Query by Boosting shows a good performance on synthetic data sets. It also generates good results on real-world data sets. However, the presence of noise in the dataset negatively affects the performance of the learning process. Again, the dataset size does not affect the performance while using Query by Boosting. The evaluation of the de-duplication function using Canopy Clustering and Bigram Indexing does not show any significant difference.

We further compare the performance results when using Query by Bagging versus Query by Boosting. First, when compare the two methods using two different blocking

methods, the experiment shows that Query by Boosting yields better results for both Canopy Clustering and Bigram Indexing. When considering synthetic versus real-world data, the same observation holds.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Herna L. Viktor, for her valuable help, advice and guidance during my graduate studies and for enhancing my interest in data mining and data warehousing. I am especially thankful for her patience, believing in me and encouraging me throughout the entire thesis project.

I would also like to thank my husband, partner, and best friend. Fabrice Nimpagaritse, for always being there when I needed him. I am deeply appreciative of his unconditional support and his encouragement throughout my thesis work.

A special thanks goes to my parents, Kazatsa Charles and Nijimbere Symphorienne. for instilling in me, since an early age, the enthusiasm of learning and inspiring me to seek higher education and to my dear sisters, Kaze Gynette, Rama Bella, Rama Lyse and Kazatsa Gratias for their moral support and for always being there for me and giving me the courage to go on in my journey.

Last but not least, I am grateful to my great friends, Isis Pena, Pauline Anthonysamy, Sepideh Ghanavati and Nadia Azam for their continued support and encouragement and their insightful advice.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Progress in technology has allowed the creation of many powerful and robust data repositories. It has also permitted the price of such repositories to dramatically drop, to the advantage of businesses and organizations. Indeed, increasingly, organizations collect, into database systems. data related to their businesses such as information about their customers, their commercial or industrial activities, their employees and so on, in order to help them to properly manage their business operations. Companies can easily find themselves in possession of many different and heterogeneous databases. The increasing use of data warehouses for reporting and data analysis, as well as the need for data integration, is an important challenge while merging different heterogeneous database systems. As a matter of fact, when merging heterogeneous databases, care has to be taken to ensure duplicates are not produced as a result of the merging process. This is especially challenging, since a common primary key is usually missing in the two databases to help us identify duplicate records that refer to the same entity. The problem of uncovering those records that refer to the same entity has been referred to in the past as *duplicate detection*, *data de-duplication*. *record linkage*, *merge/purge* and *object identification* [39, 69, 32, 5, 60, 75].

Identifying duplicate records in a dataset can be a challenging task and it should be resolved carefully. As a matter of fact, two instances referring to the same real-world entity may not be easily recognized as duplicates due to the presence of errors and inconsistencies in data. To illustrate this, let us imagine that two different databases are being merged: a dentist database and a primary care organization database. Patients, who visited both facilities, will have their demographic information stored in both databases. Sometimes, there may be cases where the information on the same patient differs in the two databases. The reason for this may be and is not limited to the following:

- Presence of typos: eg. *John* vs *Joan*

- Use of abbreviations: eg. *Street* vs *St.* or *United States* vs *USA*
- Use of nicknames: eg. *Robert McCain* vs *Bob McCain*
- Name change after marriage: eg. *Jennifer Davies* vs *Jennifer Davies-Moore*
- Use of different naming standard in different databases: eg. *Male/Female* vs *M/F* vs *0/1*

Extensive work has been conducted in the area of data mining in an attempt to deal with duplicate detection. With the invention of new classification techniques in machine learning, researchers started considering a supervised learning approach to de-duplication [30]. This approach requires that the classifier be presented with a training dataset that contains as many duplicate scenarios as possible in order to produce a better predictive model [30]. This is a tedious and expensive work that requires to manually find challenging examples of duplicates. As a solution to this problem, recent studies have focused on active learning to automate the duplicate detection process [69, 30].

Active learning is an iterative supervised learning. The learner is given control over which instances are appended to the training dataset and a domain expert is queried to determine the class label of the chosen instances. This has the effect of considerably diminishing the number of instances to label compared to the number of instances that would be needed in a regular supervised learning.

1.1 Motivation

In today's world, most companies increasingly collect and store business-related data into different repositories for later analysis. The need to merge such different and heterogeneous data sources has introduced an important problem for many companies which have to deal with the problem of data duplicates The heterogeneous characteristic of the databases makes it harder to identify those records that refer to the same real world entity.

When merging different databases for analysis, it is important that duplicates records be identified to ensure that the analysis generates accurate results. In fact, presence of duplicates may yield erroneous results. As a simple example, imagine that a company needs to know the number of customers they had in a particular month. If there exists duplicates in the "Customers" table, the resulting number would be erroneous since some customers may be counted more than once. Therefore, it is important to identify such duplicates.

This thesis focuses on the use of *Active learning* in the task of detecting duplicates in a dataset. Active Learning, in this case, is an iterative supervised learning which works by continuously teaching a classifier to distinguish between duplicates and non-duplicates using a better training dataset at each iteration [69]. Active Learning is particularly useful in circumstances where we are in presence of a large number of unlabeled data and it is expensive to find training data. In most real-world datasets with duplicates, the proportion of duplicates is usually very small compared to that of non-duplicates. Therefore, the use of active learning facilitates the task of identifying challenging duplicate records that can be used as good training examples by the classifier. It also allows the training dataset to start with a small training set with just a small number of hand-picked duplicate and non-duplicate examples.

One of the methods used by Active Learning is the use of ensemble learning algorithms to create diverse ensembles. The key idea is to create a set of models (ensembles) that are as different as possible but yet maintain consistency with the training dataset [56]. The instances that will cause most disagreement among the models will be considered as informative and will be appended to the training set.

Figure 1.1 illustrates the iterative process of Active Learning using Ensemble Learning algorithms. The process starts with a small training set containing labeled instances. Each instance has as attributes distance values between a pair of either duplicate or non-duplicate records and the last attribute represents the class label: 1 for duplicates and 0 for non-duplicates. The training set is given to an ensemble learning algorithm which creates a set of different models (a so-called "committee of hypotheses"). Few random instances from the unlabeled dataset are passed to the committee for labeling. Instances that cause the models to disagree the most on the class label are sent to a domain expert for correct labeling and are subsequently appended to the training set. These instances are considered to bring most information gain to the training set. The process is repeated until the domain expert is satisfied with the performance of the model created from the training set at a specific point in time.

The two ensemble learning algorithms considered in this thesis are "Query by Bagging" and the "Query by Boosting" [1]. Query by Bagging makes use of the "Bagging" machine learning algorithm which is a re-sampling algorithm that uses sampling with replacement on the training set in order to create a different set of instances of the same size as the initial training set. The base classifier is then run on each generated subset. As a result, different models are produced to form the ensemble of hypotheses [10]. As to the Query by Boosting, it uses the "Adaboost" algorithm to create the ensembles. Adaboost was originally created to boost the performance of the underlying classifier [70]. This algorithm maintains a certain weight for each instance in the training set. At

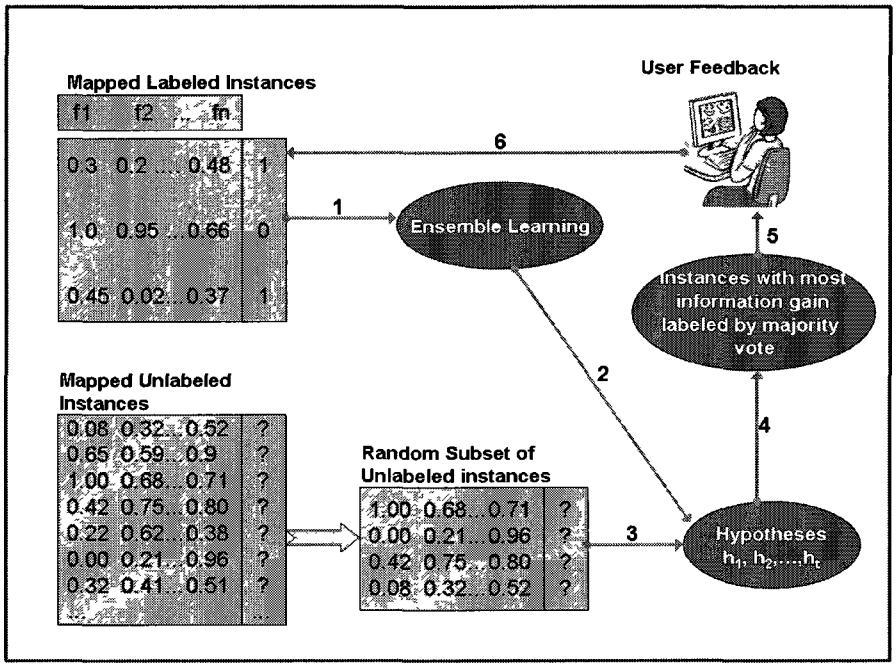


Figure 1.1: Active Learning using Ensembles

each iteration. it works by increasing the weight of the instances incorrectly classified. By doing so, it forces the classifier to focus on those misclassified instances and hence, attempts to minimize the weighted error. The ensemble is therefore constituted of the hypothesis created at each iteration.

Query by Bagging and Query by Boosting have proved to produce good ensembles which can be effectively used to identify instances that bring most information gain to a classifier. These two methods are especially recommendable for their ability to create ensembles that are diverse in the sense that they are as different as possible but yet maintain consistency with the training dataset. This feature is at the heart of Active Learning. It enables the de-duplication algorithm to quickly identify challenging examples of duplicates and non-duplicates that are used to retrain the classifier and hence produce a more powerful de-duplication function.

Some researchers have linked the de-duplication task to Active Learning. However, as far as the author is aware, no study has yet examined and evaluated how Active

Learning using Query by Bagging and Query by Boosting algorithms performed on the de-duplication task. In this thesis, we investigate the efficiency of Query by Bagging and Query by Boosting, used as Active Learning algorithms, when given the task of Data De-duplication. We also modify different parameters of the Active Learning process in order to investigate whether or not such changes have any effect on the effectiveness of the learning process. In fact, we analyze the de-duplication function using both Query by Bagging and Query by Boosting on datasets of different sizes. We also compare the de-duplication results on datasets that are synthetically generated versus real-world datasets. When performing Active Learning, blocking methods are used to minimize the cost of having to pair up all the records in the dataset by only pairing up those records that are identified as potential match and which are put in the same block. Two blocking methods are analyzed namely Canopy Clustering and Bigram Indexing. Finally, we analyze the effect of duplicate detection using Query by Bagging and Query by Boosting when the dataset contains duplicates with minor differences, such as characters swap. versus when the dataset contains noticeable differences, like very distinct values for the same attribute.

1.2 Thesis Organization

The remainder of this thesis is broken down into seven chapters. Chapter 2 provides a literature review on studies conducted in the area of duplicate detection. A review of different concepts that are utilized during duplicate detection is also provided, such as similarity functions and different blocking methods. Chapter 3 introduces data mining techniques which play an important role in the de-duplication algorithm discussed in this thesis. Chapter 4 is dedicated to Active Learning. Since this thesis focuses on Active Learning using ensemble learning, a detailed explanation of ensemble learning techniques used in the community is provided. Chapter 5 presents an experimental design which contains a detailed description of the different steps involved in the de-duplication algorithm, as well as the methods and techniques used in each step. In Chapter 6, the criteria used to evaluate the de-duplication function are first presented. De-duplication using Query by Bagging as well as de-duplication using Query by Boosting are evaluated in this chapter and the results are discussed. Chapter 7 concludes and summarizes this thesis and discusses possible future work.

Part I

LITERATURE REVIEW

Chapter 2

Duplicate Detection

Database systems are an integral part of many organizations. Data gathered in such repositories is used to perform various business operations that range from keeping track of different transactions, supporting decision making and discovering hidden knowledge for better organization of companies [31]. Hence, the management of these data repositories becomes of great importance.

The recent popularity in merging data from multiple data sources has introduced the problem of duplicate records in databases. For Instance, one can find two records that have different syntactic representation of data, but yet describe the same real world entity.

Areas such as data mining, machine learning or data warehousing, to mention a few, have found themselves weakened by the presence of those duplicates in databases [40]. Such duplicates often negatively influence the results when analyzing data. To this date, a substantial amount of research has been conducted in this field [77, 78, 37] and numerous terms have been introduced to refer to this area namely *data de-duplication*, *record linkage*, *field matching* and *duplicate detection*.

2.1 Background

Duplicate Detection focuses on discovering semantically similar records which may not necessarily be syntactically identical. A number of research activities in this area have been conducted and different methods were introduced by various researchers.

2.1.1 Work by Newcombe

The first concepts of record linkage were introduced by the geneticist Newcombe [61]. His work mainly dwells on the ability to distinguish matches from non-matches by taking into account the relative frequency at which various terms occur (among the matches and non-matches). The key idea is using this frequency value to compute a score he calls *bmit weight*, which determines whether or not two records represent a match. That is, if the two records represent the same real world entity. According to his work [60], various terms such as last names or first names with less occurring frequency (*e.g.* ***Zola***) have more discriminating power and contain important identifying information [60].

2.1.2 Work by Fellegy and Sunters

Fellegi and Sunters [32] later pursued the work started by Newcombe and developed a mathematical model for record linkage. Given two files **A** and **B**, a Cartesian product is performed, which pairs up records from both files. The ultimate goal is to classify each pair of records into a set of true match (M) or true non-match (U). In their work, they defined statistic methods that determine threshold values. The latter are used to divide the space into three categories namely the category of ***links*** (A_1), ***non-links*** (A_3) and, if not enough information is present to make a decision, the category of ***possible links*** (A_2).

First, let us consider the following ratio:

$$R = m(\gamma) \ / \ u(\gamma) = P(\gamma \in \Gamma \mid M) \ / \ P(\gamma \in \Gamma \mid M) \tag{2.1}$$

where $m(\gamma)$ and $u(\gamma)$ are the conditional probability of γ (where γ represents a set of rules representing comparison statements such as 'name is same', 'name disagree' ... and where Γ is the set of all possible realization of γ) given that a pair belongs to M and U respectively. Each rule γ_i is assigned a weight to justify the relative frequency with which certain terms occur. For instance, names such as "Zepherin" or "Zola" would have higher weights, because instances agreeing on such names have higher chances of being identical.

Next, Fellegi and Sunters identify two types of errors that may result from classification. The first error classifies an unmatched pair as a link (see equation (2.2)), whereas the second error classifies a matched pair as a non-link (see equation (2.3)). The probabilities of these errors are represented as:

$$\mu = P(A_1 \mid U) \tag{2.2}$$

$$\lambda = P(A_3 \mid M) \tag{2.3}$$

These two error probabilities will therefore be used to determine the cutoff thresholds (T_μ and T_λ). According to Fellegi and Sunters, the decision rule will classify a pair (a, b) as seen below, with the objective of maximizing the probability of positive classification (A_1 and A_3) and at the same time minimizing the probability of classifying a pair in A_2. Classifying pairs in A_2 is more expensive since human intervention is required.

- If $R \geq T_\mu$, then $(a, b) \in A_1$
- If $T_\lambda < R < T_\mu$, then $(a, b) \in A_2$
- If $R \leq T_\lambda$, then $(a, b) \in A_3$

Even though the Fellegi and Sunters theory produced promising results and have paved the way for subsequent work in this area, it presents some weaknesses. In fact, the rules γ_i used to compare records are defined by the users, which can be time consuming to produce. Also, the above mentioned method uses the conditional independence assumptions when it comes to computing $m(\gamma)$ and $u(\gamma)$, as seen in equation (2.1), in order to minimize computations. For instance, an assumption that the rules are independent of one another is made as follows:

$$\begin{aligned} &P(\text{agree fname, agree lname, agree address}|M) \\ &= P(\text{agree fname}|M)P(\text{agree lname}|M)P(\text{agree address}|M) \end{aligned} \tag{2.4}$$

2.1.3 Work by Stolfo and Hernandez

Data de-duplication later attracted the interest of other researchers who defined new methods of record linkage to adapt to specific problems. For instance, Stolfo and Hernández [39] studied how to find duplicate records in large databases. They call their de-duplication method the *merge/purge problem*. This technique makes use of the *sorted neighborhood method* which consists of arranging records in a manner such that matching records are brought close together using sort methods. First, keys are computed by concatenating attributes, or part of the attributes, with the most discriminating power. Those keys are then used to sort all the records. Next, a window of a fixed size is moved in the list of the sorted records, reducing the number of records to be compared to only those present in the window. This reduces the computational time which would be costly, had we performed a Cartesian product on all records in a large database.

To improve the performance of their method, Stolfo and Hernández propose to first cluster the data before using the sorted neighborhood and to use multiple independent runs of the sorted neighborhood method [40]. For instance, in a database of employee

records, one run can be performed using a key composed of the social insurance number (SIN), three letters of the last name and the first name. Another run may make use of a key composed of three letters of the last name first, then the SIN and finally the first name. A transitive closure is therefore done on the pairs resulting from each independent run. Stolfo and Hernández use declarative language rules they define a priori to determine the pairs that are duplicates and those that are not. Here are some examples of rules they used in their experiment:

```
Rule compare_addresses
IF compare_addresses(A.St_addr, B.St_addr)
THEN ASSERT similar_addrs;

Rule compare_st_nums
IF A.St_number AND B.St_number
AND very_close_num(A.St_number, B.St_number)
THEN ASSERT very_close_stnum;
```

When using the method proposed by Hernández and Stolfo, we need to be careful on how to choose the size of the window. In fact, bigger windows require more computational time but tend to slightly increase accuracy, while smaller windows are faster to compute but generate less accuracy. Another drawback of the method is that the declarative language rules have to be manually written and are dependent on the dataset used. New datasets require new appropriate rules. Also, the sorted neighborhood method involves a sorting phase which makes it expensive [39].

2.1.4 Work by Cohen and Richman

In 2001, William Cohen and Jacob Richman [18] described the *adaptive matching* method. The de-duplication task is given to a machine learning algorithm which learns to identify duplicate pairs from a set of potential candidate pairs. In fact, given two sources of records to be compared, a pairing process is necessary since a match or non-match is performed between every two entries. Again, it is very expensive to pair each record in the first file with each record in the second file, especially if the size of the combined file is large. To avoid this, Cohen and Richman utilize blocking methods (see Section 2.3 on page 21 for further details) which are methods used to produce subsets containing potential matching candidates and to reduce computation time.

Cohen and Richman's use of blocking methods generates subsets containing pairs with a high chance of being found to be duplicates. Each of these pairs is represented

as a vector of binary features (0,1) and the features used to encode the pairs are *edit distance, substring match, Jaccard distance, strong number match, prefix match and different types of matching tokens* [19]. Their algorithm takes a set of training examples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ where x_i is the vector of binary features and y_i is the label indicating which class the example i should belong to. A classifier then uses the training examples to learn the 'pairing function'. The latter can subsequently be used on the unlabeled dataset to predict the class label. The "Adaptive Matching" method presented by Cohen and Richman eliminates the tedious work of hand coding rules for comparison, by training a classifier to distinguish duplicates from non-duplicates. However, for the classifier to yield good results, it requires that the user picks training data that are challenging and representative of the entire dataset. This can be demanding, especially when we are dealing with a large dataset where there is a possibility that challenging duplicates can be far apart.

2.1.5 Work by Sarawagi and Bhamidipati

The work of Sarawagi and Bhamidipati follows the ideas of Cohen and Richman and delegate the de-duplication task to a machine learning algorithm. However, they go one step further and propose to make use of ***Active Learning*** in an attempt to decrease the number of training examples needed initially and at the same time increase the possibility of accurately identifying a pair of record as duplicate [69]. In fact, Active Learning eliminates the user task of having to manually choose all the training instances, which can be costly and time consuming. Instead, the learner is given control over what training instances are appended to the original small training set (around 10 instances). Active Learning is an iterative supervised learning where the classifier strategically chooses those instances that will bring more information to the classification process and from which the classifier will learn the most. Those instances are given to a user for labeling and are then appended to the training set. The learner is re-trained on the new training set and the entire process is repeated until the increased training set produces satisfactory performances. This entire process is described in table 2.1. Sarawagi and Bhamidipati work is very similar to this thesis.

Their system takes as input a database of records (D) from which we need to identify duplicates, a set of training examples (L) provided by the user (usually a small set) and a set of similarity functions (F) appropriate for each attributes. Instances in L are paired together and the similarity functions are calculated for each pair. In the end, this produces a new set of instances (L_p) having distance values as attributes and a class attribute with only two possible values to identify each instances as either a duplicate

or a non-duplicate. Similarly, a Cartesian product (DXD) on D is performed, thus providing pairs of instances which are compared using similarity functions. These pairs are unlabeled, since the class label is to be determined. If D is too large, then DXD is computationally expensive and the use of blocking techniques becomes imperative. Recall that blocking methods partition the dataset into blocks or clusters. Instances in one block are closely related to each other than instances from two different blocks. Instead of performing a Cartesian product on D, pairing is performed among instances in the same block, hence reducing significantly the cost of pairing instances. The next step is to perform Active Learning. This technique assigns most of the job of finding the most informative instance from the unlabeled pool D to the classifier. In fact, resampling techniques are used to create different predictive models. Instances that cause disagreement among the models are then considered as informative and given to a user for labeling. By prompting the domain expert (here, the user) to provide the correct label of those instances the classifier is most uncertain of, the classifier hence improves its prediction [69]. Sarawagi and Bhamidipaty show that the confusion region of the classifier will continuously decrease as informative instances are added to the training data.

---

1. Input: L, D, F
2. Create pairs L_p from the labeled data L and F.
3. Create pairs D_p from the unlabeled data D and F.
4. Initial training set $T = L_p$.
5. Loop until user satisfaction
   - Train Classifier C using T.
   - Use C to select a set S of n instances from D_p for labeling.
   - If S is empty, exit loop.
   - Collect user feedback on the labels of S.
   - Add S to T and remove S from D_p.
6. Output classifier C

Figure 2.1: The interactive de-duplication Algorithm by Sawaragi and Bhamidipaty

2.1.6 Summary

Researchers have taken different approaches with regard to identifying records that represent the same object. In this section, we have seen early work that utilized *rule-based approaches*. The problem with these approaches is that, for better results to be generated, many rules needs to be declared to catch as many anomalies and special cases as possible, which is usually time consuming. We have also seen the work of the pioneers in this field, Newcombe and Fellegi-Sunters, who took a *probabilistic approach* to record linkage, where the accumulated weight of agreement and disagreement on the fields of pairs of records is used to determine if they are duplicates or not using defined thresholds. Other researchers opted for giving the task of de-duplication to *machine learning algorithms*.

The objective of record linkage is to discern those records that points to the same real world object but which have some syntactic differences due to the presence of noises, errors and typos. Therefore, it is apparent that similarity functions, which compute how close two strings are to each other, play an important role in record linkage. The following section gives an introduction to similarity functions and describes some of the most commonly used functions.

2.2 Similarity Functions

Similarity functions are formulas that have been invented to measure the distance between two data points. In our work, the similarity functions are used to compute the degree of affinity between two values of an attribute [6]. The similarity is defined as a value indicating how close the two elements are, depending on how big or small the value is. A distinction need to be drawn here between the terms ***distance functions*** and ***similarity functions***. Given two strings s and t, a *distance function* generates a comparative value v where a bigger v represents less similarity and a smaller v represents a greater affinity between the two strings. In contrast, a *similarity function* run on two strings will generate a bigger value only if the two strings are compared to be very similar. In this thesis, the two terms will be used without distinction, unless clearly specified.

Distance functions have been used extensively in the area of data mining, machine learning, information retrieval, record linkage and biological sequence comparison [78, 17]. Multiple functions have been proposed. In duplicate detection, distance functions hold an important place and are often used directly or indirectly to determine which pairs of

instances are more likely to be duplicates.

When computing the distance value between pairs of strings. two different approaches are possible. Thus, similarity metrics are divided into two categories, depending on the approach taken. The first category considers the strings to be a sequence of characters which makes the distance value to vary with regard to the number of characters misplaced. The second category, on the other hand, considers the strings to be sets of tokens (words) where the order of the tokens is not of great importance. The more tokens two strings have in common, the more identical the strings will be.

Similarity functions in these categories are explained in the next two sections (Sections 2.2.1 and 2.2.2). Note that. most of the time, the computed distance values are normalized into a range of $[0, 1]$ to increase the ease of interpretation.

2.2.1 Similarity Functions based on Edit-Distance

Edit Distance-like functions determine the similarity between two strings of characters s and t by counting the number of edit operations required to transform a string s into a string t. A number of different algorithms have been set forth to compute edit distance functions and here are some of the most used today.

Levenshtein Edit-Distance

Vladimir Levenshtein introduced the Levenshtein Distance [51] which computes the similarity distance between two strings as the least number of edit operations. This is often done using dynamic programming. The edit operations we are referring to are namely the *insertion*, *deletion* and *substitution*. The cost of these operations is set to 1. This can be illustrated by computing a matrix containing $0 \ldots m$ rows and $0 \ldots n$ columns where m and n are the length of the strings s and t respectively. Each cell of the matrix is computed as follows:

$$D(i,j) = \min \begin{cases} D(i-1, j-1) + c(s_i, t_j) \\ D(i-1, j) + 1 \\ D(j, i-1) + 1 \end{cases} \tag{2.5}$$

where $c(s_i, t_j) = 0$ if $s_i = t_j$ and $c(s_i, t_j) = 1$ otherwise. The first line represents either a substitution or a simple copy where the value of the cost $c(s_i, t_j)$ becomes 1 or 0 respectively. The second line represents an insert and the last line a deletion. Figure 2.2 illustrates the Levenstein distance where the distance value is the value in the bottom right cell of the matrix. In our example, we compare the strings 'Divanov' and 'Diane'

		D	I	V	A	N	O	V
	0	1	2	3	4	5	6	7
D	1	0	1	2	3	4	5	6
I	2	1	0	1	2	3	4	5
A	3	2	1	1	1	2	3	4
N	4	3	2	2	2	1	2	3
E	5	4	3	3	3	2	2	***3***

Figure 2.2: The Levenshtein Distance on strings 'Divanov' and 'Diane'

and the distance between them is 3 (*i.e.* a minimum of three edit operations is required to transform the first string into the second).

Needleman-Wunch Distance

Needlemam and Wunsch [59] developed a similarity function for measuring how close two biological sequences are. This function can be considered as an improved version of the Levensthein edit function. Their function takes into account finding optimal alignment of two strings and hence allows the insertion of gaps in either strings. The cost of a gap is known as a *penalty gap*. Here is a possible alignment of the strings introduced earlier 'Divanov' and 'Diane':

```
D I V A N O V
| |   | |   |
D I —A N— E
```

Similarly to the Levenshtein edit distance, we use dynamic programming by recursively computing the cost value in each cell of a matrix representing the Needleman-Wunsch distance process. The following equation corresponds to the recursive computation of the cells:

$$D(i,j) = \min \begin{cases} D(i-1, j-1) + c(s_i, t_j) \\ D(i-1, j) + G \\ D(i, j-1) + G \end{cases} \tag{2.6}$$

where G represents the gap penalty and $c(s_i, t_j)$ is an arbitrary distance function on characters (e.g. related to typographic frequencies, amino acid substitutability, etc). In the equation, the first line illustrates the case where we align s_i with t_i, the second line

		D	I	V	A	N	O	V
	0	2	4	6	8	10	12	14
D	2	0	2	4	6	8	10	12
I	4	2	0	2	4	6	8	10
A	6	4	2	1	2	4	6	8
N	8	6	4	3	2	2	4	6
E	10	8	6	5	4	3	3	***5***

Figure 2.3: The Needleman-Wunsch Distance on strings 'Divanov' and 'Diane'

is the case where s_i is aligned with a new gap in t and the last line represents the case where t_j is aligned with a new gap in s.

Figure 2.3 illustrates the Needleman-Wunsch algorithm on the strings "Divanov" and "Diane". We set the gap penalty to $G = 2$ and $c(s_i, t_j) = 0$ if the two aligned characters are equal and $c(s_i, t_j) = 1$ otherwise. The distance value between the two strings is 5 and it is found in the bottom right corner of the matrix. This value makes sense given that we have two gaps (cost: $2 * 2 = 4$) and one substitution (cost: 1).

Smith-Waterman Distance Function

This similarity metric is also based on the basic Leveinstein edit distance. It further extends the Needleman-Wunsch algorithm and allows positive and negative cost values [74]. Instead of performing global comparison by considering each string in its entirety, this metric perform local similarity comparisons by finding pairs of substrings in the strings s and t that are similar. It also makes use of gap penalties. Each cell in the comparison matrix contains a score that represents the maximum possible score for an alignment of any length ending at that cell [74]. The Smith-Waterman final score is found in cell of the matrix with the maximum value (it can be anywhere in the matrix). The below equation illustrates the function:

$$D(i,j) = \max \begin{cases} 0 \\ D(i-1, j-1) - c(s_i, t_j) \\ D(i-1, j) - G \\ D(i, j-1) - G \end{cases} \tag{2.7}$$

Note that, usually, $G > 0$ and $c(s_i, t_j) < 0$ if $s_i = t_j$ $\left(c(s_i, t_j) > 0 \text{ otherwise }\right)$. Also, note that cells with negative values in the matrix will be replaced by 0's which will make the local alignment visible. Figure 2.4 illustrates the algorithm on the same strings.

s ="Divanov" and t ="Diane". In this example, we set the gap penalty $G = 0.5$, the cost of a mismatch $c(s_i, t_j) = 2$ (if $s_i \neq t_j$) and the cost of a match $c(s_i, t_j) = -1$ (if $s_i = t_j$).

		D	I	V	A	N	O	V
	0	0	0	0	0	0	0	0
D	0	1	0.5	0	0	0	0	0
I	0	0.5	2	1.5	1	0.5	0	0
A	0	0	1.5	1	2.5	2	1.5	1
N	0	0	1	0.5	2	***3.5***	3	2.5
E	0	0	0.5	0	1.5	3	2.5	2

Figure 2.4: The SmithWaterman Distance on strings 'Divanov' and 'Diane'

The distance value between s and t is 3.5. This means that the substrings s' = 'Divan' and t' = 'Dian' are the most similar substrings among all possible substrings from s and t. It is important to note that only one edit operation (an insertion in t' or a deletion from s') is needed for the two substrings to be identical.

Smith-Waterman-Gotoh Distance Function

Gotoh improved the Smith-Waterman algorithm by taking into account gaps of multiple sizes [36]. He introduced affine gap costs namely the cost of opening a gap o and the cost of extending a gap e. Hence, a gap of length l will have a cost of $0 + (l-1)e$. Gotoh's algorithm find the minimum cost of aligning two strings in $O(mn)$ computational time. For two strings s and t of length m and n, the comparison matrix maintains three values in each cell as follows:

$$MS(i,j) = \min \begin{cases} MS(i-1, j-1) + c(s_i, t_j) \\ IN1(i-1, j-1) + c(s_i, t_j) \\ IN2(i-1, j-1) + c(s_i, t_j) \end{cases}$$

$$IN1(i,j) = \min \begin{cases} MS(i-1, j) + o + c(+s_i) \\ IN1(i-1, j) + e + c(+s_i) \end{cases} \tag{2.8}$$

$$IN2(i,j) = \min \begin{cases} MS(i, j-1) + o + c(+t_j) \\ IN2(i, j-1) + e + c(+t_j) \end{cases}$$

$$D(s,t) = min\Big(M(m,n), IN1(m,n), IN2(m,n)\Big)$$

where $c(+s_i)$ and $c(+t_j)$ represent the cost of inserting elements s_i and t_j into the strings s and t respectively and $c(s_i, t_j)$ is the cost of matching or substituting the two characters. Note that o and e are the cost of opening a gap and extending a gap. $D(s,t)$ is the Smith-Waterman-Gotoh distance value between the two strings. The three matrices compute, for each cell, the minimum cost of edit operations between substrings of s and t up until that cell: for MS, the edit operation corresponding to that cell is a match or a substitution, for $IN1$, it is an insertion into the first string while for $IN2$, it is an insertion into the second string.

Other Interesting Distance Functions

There exists other similarity metrics based on the number of edit operations. The ***Jaro algorithm*** is such a metric and it has been encountered numerous times in the record linkage area [78]. It performs better for shorter strings and works by taking into account the number of *matches* M and then the number of *transpositions* T between two strings [43]. In fact, given two strings s and t. two characters s_i and t_j from both strings are matched if they are within a maximum distance of $\left(\frac{max(|s|,|t|)}{2}\right) - 1$. The number of transpositions is calculated by extracting the *matching* characters from both strings and aligning them. The total number of the same characters that do not line up (*i.e.* for which the character indexes are different) in the two strings is divided by two to get the number of transpositions. Hence, the Jaro formula is:

$$Jaro(s,t) = \frac{1}{3}\Big(\frac{M}{|s|} + \frac{M}{|t|} + \frac{M-T}{M}\Big) \tag{2.9}$$

Winkler modifies the Jaro algorithm so that it increases the score for the strings that present identical initial characters: in his paper [77], he sets the maximum number of initial characters that can be considered to 4. The modified algorithm is :

$$\text{Jaro-Winkler}(s,t) = Jaro(s,t) + pLength * pScale(1 - Jaro(s,t)) \tag{2.10}$$

where pLength is the longest common prefix of s and t (it is set to 4 if the common prefix length is greater than 4) and pScale is the scaling factor (set to 0.1) in Winkler's work.

Another similarity function that is worth mentioning is the ***Monge-Elkan algorithm*** [57]. This metric takes two textual fields A and B and then performs a comparison between subfields. Hence, a subfield of A will be matched with a subfield of B with which it has the highest score. The similarity score is computed as follows:

$$match(A, B) = \frac{1}{|A|} \sum_{i=1}^{|A|} max_{j=1}^{|B|} match(A_i, B_j). \tag{2.11}$$

In Monge and Elkan's paper, $match(A_i, B_j)$ corresponds to the Smith-Waterman-Gotoh algorithm (described in Section 2.2.1 on page 17) to match the subfields. This algorithm has been shown to give good similarity scores.

At this point in time, there exists a great number of string-based similarity functions. The similarity algorithms mentioned in this section are widely and commonly used in the area of record linkage and duplicate detection [78]. Cohen et. al have shown, in their article on string metric comparison [17], that most of the distance metric mentioned in this chapter, namely the Leveinshtein, Smith-Waterman, Jaro, Jaro-Winkler and the Monge-Elkan distance functions performed well on the "Name Matching" task which is a common task in duplicate detection given that many datasets used in this area contain demographic information.

2.2.2 Token-Based Similarity Functions

Token based similarity functions differ from the similarity functions based on edit operations in the sense that they consider the strings as a collection of words (or tokens). Hence, the comparison is achieved by only manipulating the tokens. The most straight forward algorithm in this category is the *Jaccard similarity* which focuses on how many tokens two strings share. The Jaccard coefficient was first introduced by Paul Jaccard [41]. It calculates the number of tokens two strings have in common over the total number of tokens (in both strings) as follows:

$$Jaccard(s, t) = \frac{|s \cap t|}{|s \cup t|} \tag{2.12}$$

A list of similarity functions based on string tokens exist to date. For instance, the *Dice Distance* function is related to the Jaccard Coefficient and gives twice the weight to agreement. It computes the similarity score by dividing the terms two strings have in common multiplied by two over the number of distinct tokens in one string plus the number of distinct tokens in the other string [24]:

$$Dice(s,t) = \frac{2\ |s \cap t|}{|s| + |t|} \tag{2.13}$$

where $|s|$ and $|t|$ represent the number of distinct terms in s and t respectively.

The most popular distance measure used in many applications is the *Euclidean distance*. Mathematically, it represents the shortest distance between two points. When used as a distance metric, it describes how far apart two string vectors are and it is defined as follows [7]:

$$Euclidean(s,t) = \sqrt{\sum_x \big(s(x) - t(x)\big)^2} \tag{2.14}$$

where $s(x)$ and $t(x)$ represent the number of occurrences of token x in the strings s and t respectively. This metric is usually used for low-dimensional data.

Another metric closely related to the Euclidean Distance is the metric called the *Manhattan distance* also known as *city block distance* or *taxi cab distance*. The choice of the name hints on the definition of this distance. In fact, graphically, the Manhattan distance represents the shortest path, in a city designed in square blocks, that a taxicab driver should follow to go from one location to another [48]. The corresponding formula is:

$$Manhattan(s,t) = \sum_x |s(x) - t(x)| \tag{2.15}$$

where $s(x)$ and $t(x)$ represent the number of occurrences of token x in the strings s and t respectively.

Both the Euclidean distance and the Manhattan distance are derived from a more generalized formula known as the *Minkowski distance* of order p (p-norm distance) [2]. It is defined as follows:

$$Minkowski(s,t) = \Big(\sum_x |s(x) - t(x)|^{\ p}\Big)^{1/p} \tag{2.16}$$

If $p = 1$, then we have the Manhattan distance (1-norm distance) and if $p = 2$. then we have the Euclidean distance (2-norm distance).

Another distance measure that is widely used to compare documents in the information retrieval community is the *Cosine Similarity* [7] which computes the affinity between two string vectors like this:

$$Cosine(s,t) = \frac{\sum_x s(x)t(x)}{\sqrt{\sum_x s(x)^2 \sum_x t(x)^2}} \tag{2.17}$$

The TF-IDF (Term Frequency/Inverse Document Frequency) is another popular function which is also often used in information retrieval [7]. It has been used many times to retrieve relevant web pages to a search query. The TF-IDF considers the frequency of a word in a document as well as the inverse of the number of documents that contain the word.

2.2.3 Summary

In summary, similarity functions are crucial for data de-duplication since we are looking to identify which entities represent the same object regardless of the syntactic errors that might be present. Similarity functions need to be chosen cautiously depending on the kind of data the experiment is applied on. For instance, if the data includes many abbreviations or if the instances show several typos or if the order of the words describing an entity does not matter. Similarity functions commonly encountered in Record Linkage have been described in this section. Let us point out that there has been many ongoing research studies in this area to improve the efficiency of these functions. For instance, recent research has focused on developing learnable string similarity functions [6, 67].

Similarity functions take as input a pair of records and output an number referring to how similar the records are. Consequently, performing a comparison between all the record pairs in a dataset D requires a Cartesian Product DXD which can be very costly, especially if the dataset is large. The following section discusses different solutions proposed to this problem called *"blocking methods"* which have been proved to minimize the number of comparison required.

2.3 Blocking Methods

De-duplication works on pairs of records in datasets. To generate all possible record pairs from two files, we would first merge them into one file D and then perform a Cartesian product DXD. This method is effective only if the size of D is not large. Since in most cases, we are dealing with large datasets, it is preferable to utilize blocking methods. Blocking methods have been developed to partition a dataset into clusters or blocks, consequently forcing the pairing task to be performed on instances from the same block [4]. Instances in one block are closely related to each other than instances from two different blocks. Note that instances that fall in different blocks are subsequently considered as non-matches.

The objective of blocking methods is to first put those records that are related in the same cluster but with a possibility of including few false matches. To be efficient,

a blocking method should have a very high recall (decrease the number of false non-matches by avoiding to classify two duplicate records in separate blocks) and a high precision (decrease the number of false matches in each block) [50]. Only instances in the same block will be paired up and later labeled as duplicates or non-duplicates.

This section introduces four popular blocking methods, namely standard blocking. sorted neighborhood blocking, bi-gram indexing and canopy clustering.

- ***Standard Blocking***: This technique was introduced by Matthew A. Jaro. It consists of assigning to the same block records that have identical *blocking key* where a blocking key is defined as a combination of one or more attributes elements [42]. The created blocks are "exhaustive" and "mutually exclusive". For instance, the records can be partitioned using the postal code as the blocking key or a combination of the first four letters of the last name and the Social Insurance Number (SIN). To deal with typographical errors in name or address spelling, Jaro proposes to use phonetic encoding such as Soundex (a phonetic algorithm used to group together similar sounding words that may have different spelling) as blocking keys. Hence, records with the same soundex code for name end up in the same blocks.

  However, this technique requires that the selection of the blocking key be performed with good care such that the chosen attributes have more discriminating power and also avoid generating blocks of big sizes. In fact. the bigger the blocks, the more the computations become expensive.

- ***Sorted Neighborhood Method***: As explained in Section 2.1, the sorted neighborhood blocking method uses sorted keys. Keys are produced by concatenating attributes or parts of the attributes and then by sorting them alphabetically. A window of fixed size w is then slid sequentially over the sorted records, thus allowing the pairing action to be performed only on the records in the same window [39].

  However. if the number of records having the same key is bigger than the size of the window, then we might loose records that would have formed perfect candidate pairs. Moreover. the presence of typos in the dataset may cause two identical records to have different keys, thus ending up in separate windows. Patrick Lehti and Peter Fankhauser proposed some modifications to the sorted neighborhood method in order to deal with its drawbacks [50]. Instead of a window of fixed size, their method uses dynamically sized windows where the size depends on the distance between the blocking keys. The distance function used between keys is a function like edit-distance.

- ***Canopy Clustering***: Canopy Clustering works by creating overlapping clusters that contain closely related records [55]. The method starts by randomly choosing a record (the center) from a set of possible candidates, *PossibleCenter*, which initially contains all the records. Similarity distance is computed between that instance and all the other instances. Only records that are at a certain distance value called *loose threshold*, T_{loose}, from the center are put in the same cluster. The center as well as the instances within a tighter threshold, T_{tight}, are removed from *PossibleCenter*. This process is repeated until the set *PossibleCenter* is empty. Finally, records from the same clusters will be paired to get the set of candidate pairs. Figure 2.5 illustrates the Canopy Clustering algorithm.

---

1. **Input:** $D = \{d_1, d_2, ..., d_n\}$, T_{tight}, T_{loose}.
2. **Output:** Set of canopy clusters $Clusters = \{C_1, C_2, ..., C_m\}$.
3. **Begin**

   $Clusters = \emptyset$, $PossibleCenters = D$.

   While $PossibleCenters \neq \emptyset$

   (a) Randomly select d in $PossibleCenters$.

   (b) Let $Canopy(d) = \{d_i \mid d_i \in D \wedge Dist(d, d_i) \leq T_{loose}\}$.
   The center d is also added to $Canopy(d)$

   (c) Let $Temp = \{d_t \mid d_t \in Canopy(d) \wedge Dist(d, d_t) \leq T_{tight}\}$

   (d) $PossibleCenters = PossibleCenters - Temp$

   (d) $Clusters = Clusters + Canopy(d)$

   End While

4. Output $Clusters$

Figure 2.5: The Canopy Clustering Algorithm

With Canopy Clustering, changing the value of the thresholds produces different outputs. In McCallum et al. paper, the thresholds parameters are tuned using a separate dataset with similar size for validation [55]. Let us assume the distance between two instances is "one" when they are very different(far apart) and "zero" when they are the same. Therefore, it has been shown that if T_{tight} decreases and T_{loose} increases, more computations will be required, since we will have many candidate pairs [55]. In fact, if T_{tight} decreases, the set $Temp$(see Figure 2.5) becomes smaller because only very similar instances are selected and if T_{loose} increases, the set $Canopy(d)$ becomes wide. This means that, only a small number of instances

will be removed from the *PossibleCenter* set at each iteration thus increasing the number of iterations and hence the number of clusters: it also means that the set $Canopy(d)$ will have many instances. In other words, decreasing T_{tight} and increasing T_{loose} will produce many clusters having many instances each. Computing candidates pairs in an such environment is very expensive.

- ***Bi-gram Indexing***: This method takes a blocking key (a concatenation of substrings of attribute values) and converts it into a list of bi-grams (i.e. substrings of two letters) [4].The user defines a threshold (between 0 and 1) which will be used to determine the number of bigrams n (picked from the blocking key bigrams) that will be used to create sub-lists of all possible combinations of n bigrams. The list of bi-grams is then sorted and inserted into an inverted index. This inverted index is therefore used to retrieve the corresponding record number in a blocks [4].

In the study conducted by Rohan Baxter et al. [4], these four blocking methods have been compared to one another using three performance metrics for blocking methods. These metrics are namely *reduction ratio* (RR) which measures the relative reduction in size of the comparison space accomplished by a blocking method, *pairs completeness* (PC) which is the ratio of the matched record pairs found in the reduced comparison space, to the total number of matched record pairs in the entire comparison space and *F score* which measures the harmonic mean of precision and recall [29]. The final results show that the two newer methods, Canopy Clustering and Bi-gram indexing, perform significantly better than the two older methods, Standard Blocking and Sorted Neighborhood method. However, for the Canopy Clustering and Bi-gram indexing, users have to carefully choose the best parameters for the thresholds to yield optimal results.

2.4 Summary

In this chapter, we have outlined the background work done in Record Linkage and we have also introduced basics concepts of similarity functions and blocking methods needed to achieve data de-duplication. Note that the Record Linkage task consists of finding data records that refer to the same entity in two or more files whereas Data Deduplication emphasizes on eliminating duplicates and therefore only keeping one copy of the entity, usually in one dataset. We provided a summary of the background work and research conducted in the area of Record Linkage. As shown in the Chapter, some research focused on probabilistic linkage, on rule-based approaches and other used machine learning algorithms. This chapter also discussed similarity functions. Finally, we

have shown few blocking methods used to reduce the number of comparisons needed to produce the set of candidate pairs.

The next chapter introduces data mining techniques that are crucial in the de-duplication algorithm presented in this thesis. The techniques include classification, ensemble learning and feature selection.

Chapter 3

Data Mining Techniques Essential In De-duplication

Data Mining methods have been employed in data de-duplication as a way of enhancing the efficiency of the de-duplication process [30]. Instead of the tedious work of manually coding rules necessary to identify duplicate records, the de-duplication process includes studying the values obtained from numerous similarity functions applied on pairs of instances. The task of de-duplication is delegated to a learner (a classifier) which will analyze the internal structure of the dataset composed of distance values and then produce a representative model highlighting the behavior of the data [69].

Research in duplicate detection has sometimes relied on supervised learning to perform de-duplication. For instance, Cohen and Richman, in their article [19], used a supervised approach to teach a learner, from a training dataset, how to cluster together records that refer to the same entity. Other researchers have used supervised learning to learn the string-edit distance functions [5, 67], and also to combine the results of different distance functions [75, 19, 5].

This chapter will introduce the concept of classification. There exists many different ways that classification may be realized. We will mainly focus on decision trees, as they are very commonly used. In fact, they are fast to train and evaluate, and they are relatively easy to interpret [47]. This chapter will also discuss advanced techniques in classification that produce an ensemble of models, instead of only one model, in order to increase the accuracy of the learner. Subsequently, the topic of feature selection will be discussed.

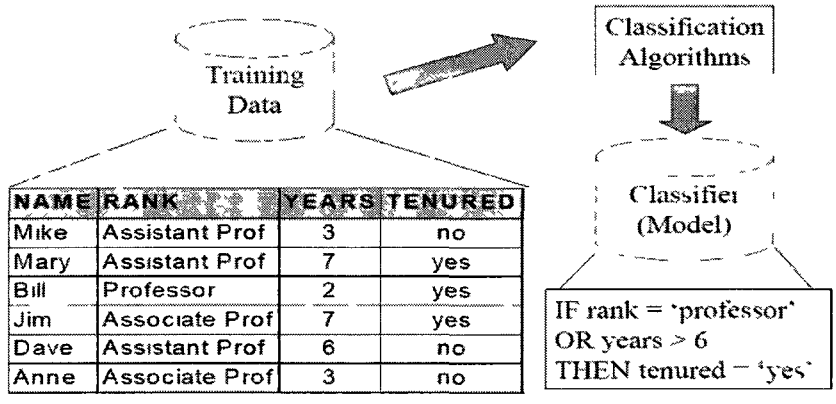
3.1 Classification

Classification is a data mining concept that consists of predicting the class to which data instances belong [38]. To achieve this goal, a classifier is first given a dataset of records, described by attributes. where one of the attributes determines the class to which the instances belong. This attribute is referred to as the **class label** of the particular instance.

The classifier learns the characteristics of the data and produces a representative model. The goal of the classifier is then to use this learned model on a new dataset of instances, where the class label is unknown, and to predict the appropriate class label [79]. The classifier is said to be a good learner if it minimizes the error in predictions. when given an independent test set of instances. Most often, the classification error is usually calculated as the number of misclassified instances. For each instance (x, y) where x is a vector of attributes and y is the predicted class label, let us call $f(x)$ the true class of the instance, then the classification error of an instance is calculated as follows:

$$error(x) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{if } y \neq f(x) \end{cases} \tag{3.1}$$

The dataset used by the classifier to produce a model is known as the **training set** and the dataset for which we need to predict the class label is referred to as the **unlabeled dataset**.



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Figure 3.1: Classification [38]

Figure 3 1 illustrates the first stage of classification. In this figure, the dataset consists of the description of faculty members and the class label identifies a faculty member as tenured or not [38]. The classifier produces a model, described here as a rule. Therefore,

the classifier makes use of the produced model to predict the class label of unlabeled instances.

Figure 3.2 illustrates the second phase of classification [38]. Using the rule created earlier by the classifier, the first instance 3.2in Figure 3.2 will be labeled as "yes" since Johnathan has a rank of Professor and Danielle, from the second instance, will not have tenure since the years in her profession are less than, or equal to, six years.

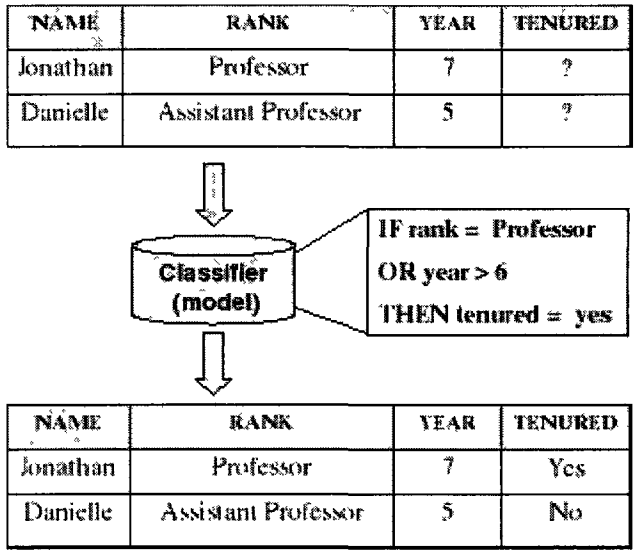


Figure 3.2: Prediction [38]

The classification process is said to be a **supervised learning** process, since the class label is given for each instance, as opposed to unsupervised learning which is a technique that do not require the class label to be defined. An example of such a technique is the well known data mining clustering technique [38].

The example presented in Figure 3.1 presented the model as a rule. Another way of doing classification is using decision trees. The following section gives an overview of classification using decision trees and presents two types of decision tree algorithms.

3.1.1 Decision Trees

Decision trees are used extensively for classification in data mining. A decision tree is a "flow-chart-like tree structure" [38] that describes trends in the underlying dataset and may be used for classification and prediction. The construction of a decision tree is a top down process, which starts with the root node and includes partitioning the data into subsets containing instances that have similar values [58]. Figure 3.3 shows one

possible decision tree for the faculty members dataset (we replaced the values of "Years" with discrete values). In the figure, the rectangles represent the attributes we split data on and the leaf nodes represent the class labels for the instances in the subset directly connected to the particular leaf node. When classifying a new unlabeled instance against a decision tree, the attributes of the instance are tested against the nodes of the tree from the root node to the leaf node which will contain the class prediction for the instance [38].

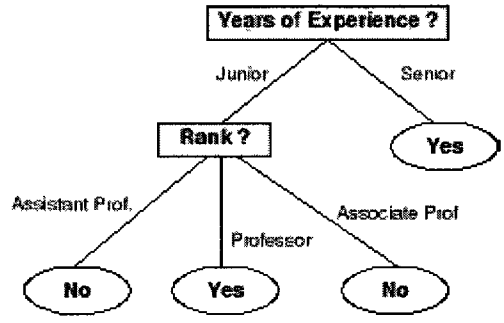


Figure 3.3: A Simple Decision Tree [38]

The C4.5 Algorithm

C4.5 was introduced by Ross Quinlan, a researcher in data mining who greatly contributed in the development of decision tree algorithms. The C4.5 algorithm creates a decision tree by iteratively creating nodes and splitting the dataset into subgroups on particular attributes of interest [63].

The C4.5 algorithm has its roots in Quinlan's primitive algorithm known as the ID3 (Iterative Dichotomiser 3) algorithm. The C4.5 classifier constructs decision trees much in a similar way as the ID3 classifier, but yet, has additional capabilities that make this algorithm capable of classifying a wide range of datasets [64].

The most important part of all decision trees is to determine how they select an attribute at each node of the tree. The C4.5 uses a measure called the *Gain Ratio* which is an improvement to the *Information Gain* (or *Entropy Reduction*) used by the ID3 algorithm. Let us first illustrate how the information gain works in ID3.

The ID3 Information Gain: The entropy reduction used by the ID3 algorithm is a concept borrowed from the information theory and it refers to the measure of disorder in a set of data. The goal will be to choose the attribute with the greatest entropy reduction (or highest information gain) to be the current node [38].

Consider a dataset S of size s with n different class labels $(C_1, C_2, \ldots, C_n)$. Let us assume that s_i represents the number of instances that belong to class C_i. The expected information needed to classify a given instance is calculated as follows:

$$I(s_1, s_2, ..., s_n) = -\sum_{i=1}^{n} p_i \; log_2 \; (p_i) \tag{3.2}$$

where $p_i = s_i/s$ represents the probability that an arbitrary instance belongs to the class C_i.

Let A be the attribute for which we would like to calculate the information gain and v the number of distinct values $(A_1, A_2, ... A_v)$ of A. We may group the instances of S into v subgroups $(S_1, S_2, ..., S_v)$ where S_j represents a subset of those instances that have the value A_j in A. If s_{ij} is the number of instances of class C_i in a subset S_j, then the entropy or the expected information required for the tree with A as a root is:

$$E(A) = \sum_{j=1}^{v} \frac{s_{1j} + s_{2j} + ... + s_{nj}}{s} \; I(s_{1j}, s_{2j}, ..., s_{nj}) \tag{3.3}$$

where the term $\frac{s_{1j}+s_{2j}+...+s_{nj}}{s}$ is defined as the weight of the j^{th} subset or tree branch [63]. The information gained by branching on A is therefore:

$$Gain(A) = I(s_1, s_2, ..., s_n) - E(A) \tag{3.4}$$

When choosing the best attribute to split the dataset on, we consider the attribute with the most reduction in entropy and therefore with the most information gain. Let us illustrate how ID3 works on a simple example. We have slightly modified the dataset shown in Figure 3.1, so that it contains only discrete values as its attributes (see Figure 3.4). Note also that the data only have two distinct values for the class attribute namely *yes* and *no* $(n = 2)$. We first compute the expected information needed to classify a certain example by applying the equation 3.2:

$$I(s_1, s_2) = I(3, 3) = -\frac{3}{6} log_2 \left(\frac{3}{6}\right) - \frac{3}{6} log_2 \left(\frac{3}{6}\right) = 1.0 \tag{3.5}$$

The next step is to calculate the entropy of each attribute. Let us start with the attribute "Rank". For each value of the this attribute, we look at the distribution of the two classes.

- value = 'Assistant Prof.' $\Rightarrow$ $s_{11} = 1 \quad s_{21} = 2 \quad \Rightarrow$
  $I(s_{11}, s_{21}) = -\left(\frac{1}{3} log_2 \left(\frac{1}{3}\right) + \frac{2}{3} log_2 \left(\frac{2}{3}\right)\right) = 0.918$
- value = 'Associate Prof.' $\Rightarrow$ $s_{12} = 1 \quad s_{22} = 2 \quad \Rightarrow \quad I(s_{12}, s_{22}) = 1$

RANK	YEARS	TENURED
Assistant Professor	junior	no
Assistant Professor	senior	yes
Professor	junior	yes
Associate Professor	senior	yes
Assistant Professor	junior	no
Associate Professor	junior	no

Figure 3.4: Small Example of a Training Dataset

- value = 'Professor' $\Rightarrow$ $s_{13} = 1$ $s_{23} = 1$ $\Rightarrow$ $I(s_{13}, s_{23}) = 0$

Applying the equation 3.3 for partitioning on the attribute "Rank", we obtain an entropy of :

$$E(Rank) = \frac{1+2}{6} I(s_{11}. s_{21}) + \frac{1+1}{6} I(s_{12}, s_{22}) + \frac{1+0}{6} I(s_{13}. s_{23}) = 0.792 \qquad (3.6)$$

Hence, the information gain for splitting on the attribute "Rank" over the entire dataset would be:

$$Gain(Rank) = I(s_1, s_2) - E(Rank) = 1.0 - 0.792 = 0.208 \qquad (3.7)$$

Doing the same thing with the attribute "Year" gives us an information gain of $Gain(Year) = 0.459$. We clearly see that splitting on attribute "Year" results in more reduction of entropy or more information gain and therefore our decision tree would first split the dataset on this attribute, therefore having the root node labeled "Year". The resulting decision tree is depicted in Figure 3.3.

The information gain measure used in the ID3 algorithm to determine the most informative attribute to split the dataset on, revealed to favor attributes with many distinct values. For instance, if the attribute under consideration contains different values in each record, then the value of $E(A)$ is 0 which will erroneously maximize $Gain(A)$. For this reason, Quinlan decided to improve this measure in C4.5 and proposed to use a measure called *Gain Ratio* to compensate for the shortcomings of the ID3 Information Gain.

The C4.5 Gain Ratio :

The bias depicted in the Information Gain used by the ID3 is rectified in C4.5 by the Gain Ratio measure. This measure applies some kind of normalization which adjusts the

tendency of the Information Gain measure to favor attributes with more distinct values [64]. This Gain Ratio measure is defined as follows:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \tag{3.8}$$

where SplitInfo represents the information due to partitioning the dataset S into v subsets $S_1, S_2, ..., S_v$ and each subset S_j contains those instances that have the value A_j of attribute A:

$$SplitInfo(A) = -\sum_{j=1}^{v} \frac{S_j}{S} \, log_2 \left(\frac{S_j}{S}\right) \tag{3.9}$$

The *Split Information* tends to increase as the number of an attribute outcome increases which therefore reduces the GainRatio to a reasonable number [64].

The Improvement of C4.5 over ID3 :

The C4.5 decision tree was developed as an improvement to the primitive ID3 algorithm. The ID3 is a very basic algorithm and presents a number of limitations. In fact, the ID3 algorithm works on training sets that only have categorical or discrete-valued attributes. Moreover, this algorithms does not deal with datasets containing missing values which limits the number of datasets the algorithm may work on. Further, when constructing a decision tree with ID3, the resulting tree always attempts to extend each branch as far as it may go to correctly classify the instances pertaining to that branch. This implies that some branches may reflect anomalies in the dataset such as outliers and noise [38] and hence cause **overfitting**. This is a common problem encountered in decision trees. The solution to this problem is called **pruning** which consists of replacing a subtree with just a leaf where necessary. Pruning may be done either early on, during construction, by stopping to split the dataset at a certain node (*pre-pruning*) or later, after the construction of the tree, by getting rid of particular subtrees.

The C4.5 decision tree algorithm was developed to take care of the problems just mentioned above.

The C4.5 algorithm provides enhancements enabling it to take care of continuous attributes. For instance, we could add the income of the faculty members as an attribute to the dataset presented earlier. Therefore, splitting on this attribute (let us call it attribute A) involves finding a suitable threshold that will divide the dataset into those instances with the income values less than or equal to the threshold and those with the income values greater than the threshold. To do this, the C4.5 algorithm first sorts the instances on the values of this attribute. If this numeric attribute has m distinct

values $(v_1, v_2. .... v_m)$, we would have $m - 1$ possible thresholds we may use to split the dataset. In fact, the midpoint between any value v_i and v_{i+1} splits the dataset into two subgroups and the threshold is defined to be the biggest value in A that does not exceed that midpoint. Hence, for each of these thresholds, we calculate the information gain as explained in the previous section and choose the threshold that maximizes the gain [64].

C4.5 can handle datasets containing attributes with missing values. When constructing the C4.5 decision tree, Quinlan [64] proposes that we compute the information gain for an attribute A by only taking into account those instances that have known values. In such a case, the value of $I(s_1, ... s_m)$ will decrease since we use less instances due to missing values and the split information $E(A)$ will increase given that we have an extra attribute value ('unknown'). The constructed decision tree keeps track of the number of instances that fall in each leaf. Note that an instance with a missing value for an attribute A will be added to all the branches containing the possible values for the missing value, therefore increasing the number representing the instances in each leaf by a certain weight. When classifying a new instance with missing values against the built decision tree, we compute the probability of all the possible results.

C4.5 deals with the problem of over-fitting the data by avoiding the construction of very complex trees with many partitions. It uses the pruning method that gets rid of parts of the tree that do not contribute to classification accuracy of unlabeled instances [64]. This has the effect of producing a more comprehensive decision tree.

The CART Algorithm

The CART algorithm is another classification algorithm which was introduced by a group of four famous UC Berkeley and Stanford statisticians namely Leo Breiman, Jerome Friedman, Richard Olshen and Charles Stone [13]. They named their decision tree algorithm **CART** which is an acronym that stands for *Classification and Regression Trees.*

The construction of the CART decision tree bears a similarity to the construction of the C4.5 algorithm. Indeed, similarly to C4.5, the CART tree is built by recursively partitioning each node into child nodes. However, the CART algorithm only splits each non-leaf node into two child nodes and is therefore called a binary tree. The process of growing the CART decision tree starts from the top node also known as the root node using repeatedly the following steps until a stopping criterion is encountered [13]:

1. First, the algorithm finds the best split for each attribute.
   To do so, the algorithm examines all possible splits and uses a splitting criterion (which we will explain shortly) to determine the split that will maximize the criterion. It proceeds differently depending on whether the attribute has categorical

values or continuous values. For the case of continuous values, the algorithm first sorts the attribute values (or split points) in an ascending order and then analyzes each value from the top down, to find the best splits that maximize the splitting criterion. At each split point, let us call it a, the algorithm checks all the instances to see if they satisfy a certain condition. The answer to the question of whether the instance at hand satisfies the condition or not determines which child node the instance will go to. For instance:
if $x \leq a$, the case goes to the left child node, otherwise, it goes to the right child node.
For the case the attribute is categorical, the algorithm analyzes each possible subset of the attribute values. Let us call the subset S. The algorithm uses the subset in a condition formulated in a similar manner as follows:
if $x \in S$, the instance goes to the left, otherwise, it goes to the right.

2. Next, among the best split points found in the previous step, the algorithm chooses the one with the highest splitting criterion.

3. Finally, the splitting of the node is performed according to the best split found in the second step.

The CART algorithm repeatedly attempts to break each node into two smaller child nodes, the goal being to have child nodes that are "purer" than their parent nodes. Hence, some functions referred to as *"impurity functions"* provide a way to measure the impurity of nodes in decision trees. The most commonly used impurity function in the CART algorithm is the *"Gini measure"*. It is in the same category as the "entropy measure" used by the C4.5 decision tree.

Consider a set of instances S in the node v with n number of instances. Each instance is assigned a class label j from all possible c classes. The Gini impurity function at a node v is defined as [13]:

$$Gini(v) = 1 - \sum_{j=1}^{c} \left(p(j|v) \right)^2 \tag{3.10}$$

where $p(j|v)$ is the probability that an instance has a class j given that it falls into the node v. In other words, $p(j|v)$ represents the proportion of instances with j as their class label in the node v.
The Gini impurity function may be interpreted as being the probability an instance at the node is classified incorrectly [13].

When the node v is partitioned into two child nodes v_1 and v_2, then the Gini index of the partitioned data is calculated as follows:

$$Gini(a, v) = \frac{n_1}{n} Gini(v_1) + \frac{n_2}{n} Gini(v_2) \tag{3.11}$$

where n_1 and n_2 are the number of instances in v_1 and v_2 respectively and a is the split point. Hence, *"the splitting criterion"* also referred to as **"the decrease of impurity"** or **"the goodness of the split"** is defined as:

$$\Delta Gini(a, v) = Gini(v) - Gini(a, v) \tag{3.12}$$

The Splitting Criterion is computed for all possible split points. In the end, the attribute value that maximizes the splitting criterion will be used to further split the parent node. During the construction of the CART tree, the algorithm verifies at each node if it should stop the construction process or continue splitting the node. The construction of the tree is stopped when one of the following conditions is met:

- the node is pure and all the instances are in the same class
- the splitting results in only one instance in the child nodes.
- the attribute values of all the records are identical.

If no tree growing constraint is specified (such as a user-specified limit on the number of levels in the tree or a user-specified minimum size value of nodes), the construction of a cart decision tree can grow large until the final leaf nodes contain only one instance. In such a case, the tree may be over-fit and represent features of the present dataset that might not be found in future test data. The solution to this problem is to prune the tree by getting rid of the subtrees that do not contribute much to the overall accuracy of the tree. For the CART algorithm, Breiman proposes the use of *minimal cost-complexity pruning* which uses the post-pruning approach (the tree is first grown as large as possible before pruning). It attempts to find the best pruned subtree with respect to the misclassification cost [13]. A good way to estimate the error of misclassification in the tree is by using the V-fold cross validation, especially since finding a separate large dataset to estimate the error rate can be hard.

Pruning using the V-fold cross validation has been shown to be effective [13] and it determines how much to prune from the original maximal tree by creating a sequence of cross validation trees. The error rates of those trees are mapped to the nodes of the original maximal tree to estimate its error rate. Here is how it works: Initially, an optimal tree is constructed using the entire dataset. This dataset is then divided into V mutually exclusive subsets referred to as folds (usually 10 folds). This is accomplished using

stratification which ensures that the distribution of classes in the subsets is relatively similar to that in the initial dataset.

The next step puts aside one subset to be used as the test data. The remaining subsets, which are combined to form the new training data, are used to build a tree. The subset that was held back is run on the tree to get the class prediction; the classification error for that subset is computed. We repeat this process by iteratively holding back one subset (a different set each time) as the test set and leaving all the others to be used as the training set. This process is stopped when all the folds have been used as test sets. At the end of the iteration, we will have V different trees which we refer to as test trees.

The error rates of the test trees are combined and mapped to the nodes of the original maximal tree. In other words, the test tree error rates will be used to estimate the error of the maximal tree nodes. Once we have the estimated error of each node in the maximal tree, the pruning can hence be performed and a optimal pruned tree found. Instead of choosing the minimal error tree, Breiman proposes to use One Standard Error Rule(1 SE Rule) which chooses the smallest tree such that its error is at most one standard error bigger than the minimal tree.

The CART algorithm is also capable of taking care of missing values. Earlier, we saw that each node in the CART tree is split according to the attribute that ensures maximal purity of the child node. When classifying an instance with the value of the splitting attribute missing, the CART algorithm chooses a surrogate splitting value. The surrogate splitter is used in place of the missing primary splitter and behaves in a similar manner as the primary splitter in regards to the class prediction. Hence, CART chooses the best splitter available using the association rule to compensate the missing primary splitter.

Discussion

It is important to mention that there exists extensions to the decision trees C4.5 and CART that have been developed to add new capabilities to the decision trees. A few research studies have adapted the C4.5 and CART algorithms to deal with *multi-label classification* [16, 15, 9] which is a type of classification where the instances may have more than one class they belong to. Multi-label classification is most commonly used in the text classification, where a document may be relevant to multiple topics and in functional genomics, where genes may have multiple functions [8]. For instance, Clare and King [15] adapted the C4.5 decision tree algorithm to deal with multi-label classification by modifying entropy computation. Moreover, techniques have been proposed to handle very large datasets. External memory algorithms as well as parallel implementations have

been proposed to improve decision trees such as C4.5 and CART in order to handle very large datasets [35, 44, 73]. Such algorithms include namely the RainForest framework, SPRINT and ScalParC algorithms.

Classification, as explained in this section, produces a model or a hypothesis with the objective of minimizing the prediction error when assigning class labels to new unlabeled instances. Recently, in machine learning, a number of methods that attempt to increase the classification accuracy have been proposed in the area of supervised learning. *Ensemble Learning* is such a method and is explained in the subsequent section.

3.2 Ensemble Learning

In contrast to traditional classification that finds one best hypothesis to describe the data. *Ensemble Learning* creates a set of models or hypotheses (also referred to as "committee" or "ensemble"). Many different ways have been proposed with regards to creating a set of different hypotheses. but we will mainly focus on one common approach of resampling the training dataset. This technique runs the learning algorithm (also called the *base learner*) several times, and at each iteration, the learner is given a different subset of the training example [25]. Once the committee is created, the class label of a new instance is predicted by combining the votes of each individual hypothesis in the ensemble, in a certain fashion (typically by weighted or unweighted voting) [25].

Ensemble learning is often known to increase the accuracy of the base learner. In fact, when combining the decisions of multiple hypotheses, each of which has an error rate below 0.5 (at least more accurate than random guessing). and where the errors are somewhat uncorrelated, the disagreement between the hypothesis will "cancel out" and correct decisions will be strengthen [28]. Therefore, constructing a good learner requires that the hypotheses be diverse but yet maintain the consistency within the dataset.

The *Bagging algorithm* [10], the *Boosting algorithm* as well as the *Stacking algorithm* [33] are methods that are commonly used to create a committee of hypotheses. They have shown to significantly increase the accuracy of the base learner. These methods draw their popularity in the fact that they accept any learning algorithm as their base learner. This has the advantage of giving the user the option of choosing the appropriate learning algorithm to boost, that is appropriate to a given domain. For this reason, Bagging, Boosting and Stacking are referred to as *meta-learners*. The following two sections briefly explain how meta-learners construct a model.

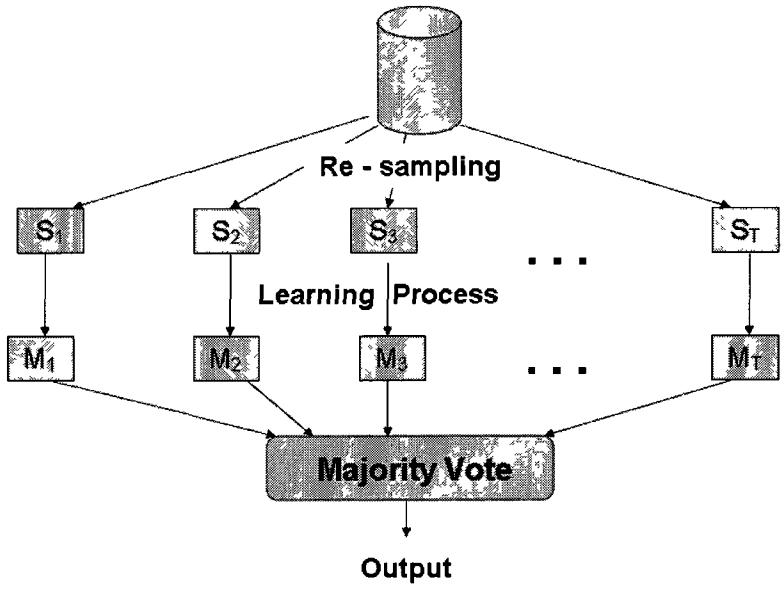


Figure 3.5: The Bagging Process

3.2.1 Bagging

The Bootstrap Aggregating (or Bagging) [10] algorithm makes use of the bootstrap resampling method which randomly draws m instances from the original dataset with replacement. Figure 3.5 illustrates how Bagging is conducted. The algorithm first takes a dataset S and creates different subsets $\{S_1, S_2, S_3...S_t\}$ by randomly picking instances from S with replacement. The resulting subsets are referred to as the *bootstrap replicates* of the original dataset and they are of the same size as the original dataset S with some instances being represented more than one time. It has been shown mathematically that around 0.368 of the original dataset is not selected and therefore used as test data [82].

Let us assume that we want to construct an ensemble of T hypotheses. Bagging performs bootstrapping T times to create T different versions of the original dataset $\{S_1, S_2, .. , S_t\}$ The base learner is therefore applied to each bootstrap replicate to finally produce an ensemble of T hypothesis $\{M_1, M_2, .., M_t\}$. Prediction on new instances is done by performing majority voting of all the hypotheses.

Bagging performs better with learning algorithms that are not stable, i.e, learning algorithms for which a small change in the training data results in a significant change in the output hypothesis [28]. Breiman, in his article [12], shows that subset selection, neural networks and classification and regression trees are unstable whereas the k-nearest neighbor methods are stable. Bagging exploits the instability of the learner to improve

classification accuracy [26]. In fact, the instability of the base learner allows the formation of ensembles that are diverse. The majority vote performed on these ensembles will ensure an improved accuracy of the learner. On the other hand, Bagging does not improve accuracy if the base learner is stable [28]. It was also shown that Bagging still performed well in presence of noise [26].

3.2.2 Boosting

The boosting algorithm is another method used to create an ensemble of hypotheses to boost the performance of the underlying base learner also referred to as the "weak" learning algorithm [33]. This algorithm maintains a certain weight for each instance in the training set. At each iteration, boosting attempts to minimize the weighted error. In fact, boosting increases the weight of instances incorrectly classified by the previous hypothesis. This forces the learner to focus on those misclassified instances during the next learning process. This process is repeated for a specified number of iterations.

There exists two ways the boosting algorithm manipulates the weights in the original dataset (let us call it set S) to produce a new training set to be fed to the learning algorithm [26]. The first approach, *boosting by sampling*, generates a new training set by sampling with replacement from S with the probability of selection of instances being proportional to their weight. The second approach, *boosting by weighting*, feeds the entire dataset S to a base learner that supports a weighted training set and uses the weights during the construction of the model. We focus on a version of boosting by weighing called "Adaboost.M1". Figure 3.6 provides the algorithm for Adaboost.M1.

Adaboost.M1 works as explained earlier by taking into account the weight distribution of the instances in the training set. After each iteration t, Adaboost.M1 measures the error of the current hypothesis and if the error is greater than 0.5, the iterations are terminated and the total number of hypotheses generated is $t-1$. Otherwise, the weights of the instances are adjusted for the next iteration by multiplying the current weight of the correctly classified instances with a factor β_t which is a function of the error (see details of the algorithm in Figure 3.6). This has the effect of decreasing the weight of correctly classified instances and hence highlighting the weight of misclassified instances. The normalization process that follows ensures that the sum of the adjusted weight is equal to one. The final boosted hypothesis is the weighted vote of each individual hypothesis. The weight of each hypothesis is based on its accuracy on the corresponding weighted training set it was trained on.

The boosting algorithm is fast and efficient [70]. It is a meta learner that may be used with any base learner, even those that only provide moderately accurate classifier.

The base learner only needs to be better than random. It has been shown that Adaboost has the ability to identify noise [70]. In fact, given that at each iteration the weight of the difficult examples is increased, the instances with the highest weight usually end up being outliers. However, boosting will perform poorly if given insufficient data. Similarly to bagging, boosting also works better if the base learner is unstable.

Pseudocode: Adaboost.M1

Input:
S - Set of m samples: $S = [(x_1, y_1), \ldots, (x_m, y_m)]$
A - Weak learning algorithm
T - Number of boosting iterations
Initialization: $\forall i \leq m, D_1(x_i) = \frac{1}{m}$
For $t = 1, \ldots, T$

1. Run A given the distribution D_t
2. Let us name its output hypothesis h_t
3. Compute the error rate ϵ_t as follows:
   $\epsilon_t = \sum_{h_t(x_i) \neq y_i} D_t(x_i)$
4. If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop
5. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$
6. Update the weights
   $D_{t+1}(x_i) = \begin{cases} \frac{D_t(x_i)\,\beta_t}{Z} & \text{if } h_t(x_i) = y_i \\ D_t(x_i) & \text{otherwise} \end{cases}$
   (Z is a normalization constant such that $\sum_{i=1,\ m} D_{t+1}(x_i) = 1$)

Output: The final hypothesis:
$h_{fin}(x) = \arg\max_{y \in Y} \sum_{h_t(x)=y} \log \frac{1}{\beta_t}$

Figure 3.6: The Adaboost.M1 Algorithm

3.2.3 Stacking

Thus far, we have considered *homogeneous* ensemble learning algorithms (Bagging and Boosting) that employ only one learning algorithm to create diverse ensembles. The Stacking algorithm is another ensemble learning algorithm that may be used with a single learning algorithm. However, Stacking is most commonly used to generate diverse ensembles using different learning algorithms. In this case, the output of these multiple base classifiers will be combined using another learning algorithm, referred to as a meta-level classifier [81].

Stacking, also known as *Stacked Generalization*, was introduced by David Wolpert [81]. Stacking applies diverse learning algorithms on one dataset partitioned in a certain way, as we will see shortly. But first, let us review some notations that will be used:

S - the dataset
s_i - the b^th sample represented as (x_i, y_i) where
  x_i - vector of attribute values and
  y_i - class label
m - number of samples
L_k - the k^th learning algorithm
N - number of learning algorithms
r - number of partitions
P_{ij} - a set of partitions where
  i is such that $1 \leq i \leq r$ and
  j is such that $j \in \{1, 2\}$

Stacked Generalization may be divided into two phases. The first phase consists of applying the N learning algorithms or base-level classifiers on the partitioned dataset while the second phase consists of combining the outputs of the base classifiers using another learning algorithm referred to as a meta-level classifier [76].

In the first phase, the dataset is first partitioned using cross-validation. Wolpert proposes to use the "Leave-One-Out Cross Validation" (LOO-CV) which splits the dataset S into r partitions. Each iteration of the LOO-CV produces one partition and each partition contains a training set and a testing set: $Partition_i = (P_{i1}, P_{i2})$. In this case, the testing set P_{i2} consists of a single element while the training set P_{i1} contains the remaining elements. The LOO-CV process is repeated in such a way that each data instance is used as the testing data. Therefore. the number of partitions is the same as the number of sample instances ($r = m$). Subsequently, we train all the N base learners using the training set P_{i1}, for $i = 1 \ldots . r$ and then use the instance in the test set P_{i2} as a query for the learning algorithms. Let us refer to the prediction on that query by the k^{th} learning algorithm as $\widehat{y}_i^k$.

In the second phase, we create a meta-level dataset using the predictions made by the base classifiers. This new dataset contains examples of the form $\left((\widehat{y}_i^1, \ldots . \widehat{y}_i^N), y_i\right)$ where the features (or attributes) are the predictions of the base-level classifiers and the class is the actual class of the query example (in the test set) at the i^{th} partition.

To answer a base level query of the form $(x_b, ?)$ where x_b is a vector of features and ? is the sought class label, this query goes through all the steps mentioned above. In fact, the

N learning algorithms make predictions on this instance after being trained, this time, on the entire dataset S. A meta-level instance without a class label is therefore created using the N predictions as its features. The newly constructed meta-level instance will be used as a query to the meta-level classifier. In order to answer the query, the meta-level classifier is first trained with the meta-level dataset and the answer to the query will be the answer to the base level query [81].

Stacked generalization is an example of an ensemble learning algorithm which uses a meta-level classifier in order to combine guesses from multiple base-level learning algorithms. The objective of stacked generalization is to reduce the generalization error [81] and hence improve the overall predictive accuracy. It has been shown that stacking increases the accuracy with respect to the base classifiers.

3.3 Feature Selection

When gathering data for future analysis, the tendency is to get as much information describing the objects in the records as possible. The reason for this is that it is difficult to predict "a priori" what features will be more appropriate than others for different types of analysis that will be conducted in the future [23]. For instance, during classification, some attributes might be redundant in the sense that they do not add anything to the model or they might be irrelevant if they do not affect the model in any way [23], therefore the need to perform what is called *feature selection*. Dash and Liu [23] provide a good explanation of how feature selection works. We will abide by their definition.

Given a set of candidate features (or attributes), the goal of feature selection is to select the smallest subset of features such that the classification accuracy does not significantly change and at the same time the class distribution remains as close as possible to the original class distribution before removing irrelevant features [23].

In theory, feature selection picks, among all possible 2^n subsets of features, the optimal subset that yields to a robust learning model. However, this requires an exhaustive search which may be computationally expensive, even for a dataset with a moderate number of n features. Proposed solutions to this problem are to utilize heuristic or random search methods to decrease the complexity of the exhaustive search. Dash and Liu divide any typical feature selection method into four basic steps:

The search procedure: This step is also known as the "generation procedure". Its aim is to generate the next candidate set. This procedure starts either with no features and at each iteration, new features are appended to the candidate set (*forward selection*) or it starts with all the features and iteratively features are

removed from the candidate set (*backward elimination*) or else it starts with a random subset of features.

The evaluation function: This step measures how good the feature subset generated by the search procedure is and then compares it to the previous best feature subset. If it is better, it then becomes the current best subset of features.

stopping criterion: This step is necessary especially because we are avoiding the complexity of performing an exhaustive search to find the best subset. The iteration will continue until the specified criterion is satisfied and the output will be the current best subset at the time the stopping criterion is reached. Hence, the stopping criterion may be used during the search procedure where it checks whether a certain number of features is selected or if we have reached a certain number of iterations. The stopping criterion may also be based on the evaluation function. In this case, the criterion might check whether adding or deleting a feature to or from the subset improves the robustness of the subset or it might check if the best subset is obtained based on some evaluation function.

validation: The validation step takes as input the proposed best subset of features and performs some tests to assess the validity of the selected subset. Dash and Liu argues that this is a separate step but an important step nonetheless.

Algorithms used for feature selection may be classified into two categories, the *wrapper model* and the *filter model.* The wrapper subset evaluation method analyzes subsets of attributes with regard to their usefulness to the class label. It performs a search of a good attribute subset by using the learning algorithm itself as part of the function evaluating feature subsets [46]. The learning algorithm estimates the accuracy of the training data using cross-validation, with different sets of features removed from the dataset. The feature set with the highest evaluation is chosen as the final set [46]. The filter method, on the other hand, performs feature selection as a pre-processing step without using a learning algorithm. The general characteristics of the training data are utilized to select relevant features [68].

The most important advantage of the wrapper methods is that it provides better results than the filter methods with regard to finding attributes that are best fitted for the learning algorithm [22]. With the filter model, the issue is that it ignores the effect of the selected feature subset on the performance of the learning algorithm [46]. Also, the wrapper algorithm is computationally more expensive due to the fact that it has to call the learning algorithm for each feature set considered [49].

Feature selection plays an important role in data mining and machine learning. By removing the unnecessary attributes, this technique generates a new dataset with a smaller feature dimensionality that may greatly influence the speed of learning algorithms as well as the robustness of learning algorithms. Feature selection is often used in an attempt to decrease the problems associated with dataset with high feature dimension.

3.4 Summary

In this chapter, we covered the main techniques in data mining that are used to improve the efficiency of de-duplication algorithms. The concept of classification has been explained, where a dataset is given to a learner and a mapping is done between the object features and the class labels. In the case of de-duplication, the learner is given a dataset of instances where the attributes represent a numerical value describing the distance between two particular instances. The class attribute is a binary set $\{0, 1\}$ where an instance with class label 1 means that the represented records are duplicates and 0 means they are non-duplicates. The classification will learn to predict if new instances are duplicates or not. This chapter also explained how ensemble learning works to improve the performance of the base learner. Three common algorithms for producing ensembles namely bagging, boosting and stacking were presented. As we will see later in the following Chapter, our de-duplication function will use Active Learning with ensemble learning to select informative instances that will greatly contribute to the creation of an efficient de-duplication function. Finally, we discussed the concept of attribute selection which consists of removing those attributes that do not contribute much in the learning process. During the de-duplication process, attribute selection will help us to only focus on a smaller group of attributes that is relevant to classification, thus improving the speed of the learning process.

In the following chapter, we introduce the concept of Active Learning, a crucial concept in the algorithm we are using to perform de-duplication. The chapter will describe and cover the main work done in this area as well as commonly used algorithms to carry out Active Learning.

Chapter 4

Active Learning

In data mining, a classifier takes as input a training dataset and produces a model that may later be used to predict the class label of new instances [79]. Classification may also be defined as "learning by examples". The challenging question is: In presence of a large dataset of unlabeled instances and a relatively small set of training data, how can classification be improved so that the generalization error (error on future unseen data) is reduced ?

In response to this question, which is of paramount importance in data mining, a very good method has been introduced and has been proved to be up to the challenge [20, 21, 53, 71]: *"Active Learning"*. Active Learning, in our case performed using *"resampling"*. focuses on giving the learner (classifier) some control over what training data to use [53]. The classifier particularly chooses those instances that will bring more information to the classification process and from which the classifier will learn the most, thus requiring less data to produce a good model. In the subsequent sections, we will review the definition of Active Learning and then present three different algorithms used when performing Active Learning namely *query-by-committee*, *query-by-bagging* and *query-by-boosting*. These three algorithms perform Active Learning using ensembles generated by an ensemble learning algorithm as we will see later in this chapter.

4.1 Active Learning through Ensemble Learning

Active Learning is an iterative process which consists of strategically choosing an instance (or a small set of instances) from a pool of unlabeled instances and acquiring the right class label(s) from a domain expert; once the class label is obtained, the instance(s) is (are) appended to the previous training dataset and the learner is re-trained, this time with the incremented training data and a new model is generated. The process is

repeated until the performance of the model obtained produces satisfactory results.

The success of this method relies on the way the classifier chooses the unlabeled instances which, when provided with their true class label, will bring most information to the training dataset. One innovative technique involves using ***"uncertainty"*** as way of identifying those instances with utmost information. In fact, the chosen instances for labeling are those for which the learner is most uncertain of the class labels regardless of the previous labeled instances [52].

An efficient way of depicting the uncertainty is by making use of "Ensemble Learning", as explained in Section 3.2, to create diverse ensembles. The key idea is to create a set of models (ensembles) that are as different as possible but yet maintain consistency with the training dataset [56] and then focus on those instances for which the disagreement amongst the ensembles is the greatest. Appending those targeted informative instances to the existent training set and then re-training the classifier is expected to create a model with a better accuracy than the previously produced model, as well as improve the performance of the classifier on future unlabeled data [20, 45, 52].

The following sections illustrate three very efficient algorithms namely *Query by Bagging*, *Query by Boosting* and *Query by Committee*. These algorithms have been used in the past to perform Active Learning and they respectively use the following resampling algorithms: Bagging, Boosting and a randomization algorithm. These three algorithms perform Active Learning in much the same way by creating ensembles and focusing on those instances that cause the most disagreement among ensembles. However, they differ in the way they resample the training set to create the committees.

4.2 Query By Committee

The concept of Query by Committee (QBC) was introduced in the article by Seung et al. [72]. The authors show that giving control to the learner over which instances to train from, decreases exponentially the generalization error. The Query by Committee works by creating a committee of n hypotheses (usually two hypotheses). To do so, it relies on a sampling algorithm known as "the Gibbs" algorithm which is a randomized algorithm that selects a hypothesis from a given version space[1] according to the posterior distribution [72].

The Gibbs algorithm is run n times to produce n hypotheses. Let us consider a case where the Gibbs algorithm is run two times. Let the two produced hypotheses predict the labels of the unlabeled data. The QBC chooses the instance for which the two hypotheses

[1]The version space is the set of hypotheses that are consistent with the previous training sample

Algorithm 1 Query by Committee

Input: The following are required:

N : Number of iterations

L : The randomized Gibbs algorithm

T . Number of time the Gibbs algorithm is called

S : Number of randomly picked unlabeled instances

U . Set of unlabeled data

Initialization: Small training data $D_1 = (x_1, y_1)$

1 **for** $i = 1, 2, \ldots N$ **do**

2 Run L on D_i T times to produce $h_1, h_2, \ldots, h_T$

3 Randomly generate a small set of S points from U with respect to uniform distribution over U. We call the set C.

4. Choose an instance $x^* \in C$ such that it brings most disagreement among hypotheses
$x = arg\ min_{x^*} | \{t \leq T \mid h_t(x^*) = 0\} - \{t \leq T \mid h_t(x^*) = 1\} |$

5. Provide the instance x^* to a domain expert for labeling and obtain the label y^*

6. Append the newly labeled instance (x^*, y^*) to the training set:
$D_{i+1} = D_i + \{(x^*, y^*)\}$

7 **end for**

Output: The final hypothesis is as follows:

$h_{fin}(x) = arg\ max_y | \{t \leq T \mid h_t(x) = y\}$

where h_t are hypotheses obtained during the last iteration.

disagree on the class label and then asks a domain expert to provide the true class label. Adding this newly labeled instance to the existing training sample is believed to bring more information to the learning process than if the instance was randomly picked and labeled from the unlabeled set.

The pseudocode for the QBC algorithm is provided in Algorithm 1. The algorithm depicts a case where the Gibbs algorithm is run n times. For the case where $n > 2$, the disagreement is defined as the case where the hypotheses are split most evenly with regards to the assignment of the class label. Note also that the most informative instance is drawn from a smaller unlabeled subset with the same uniform distribution as the entire unlabeled set. This prevent us from having to predict the class label of all the unlabeled instances in order to choose the one that causes most disagreement among hypotheses.

Freund et al. [34] have demonstrated that the Query by Committee reduces significantly the number of training instances necessary to produce an efficient model. Also, they show that under certain assumptions, the prediction error decreases as new informative instances are added to the training dataset. However, the choice of using the Gibbs introduces computational complexity. This is due to the fact that the Gibbs algorithm is intractable [34]. In an attempt to deal with this issue, two algorithms where introduced by Abe and Mamitsuka [1]: Query by Bagging and Query by Boosting. These two learning algorithms respectively rely on two resampling methods namely Bagging and Boosting, to generate the committee of hypotheses. The following sections describe how they work.

4.3 Query By Bagging

The Query by Bagging takes its origins in the Query by Committee. Just like the QBC, the Query by Bagging iteratively picks an instance from a set of unlabeled samples which it believes will bring the most information gain to the training set (initially very small) as opposed to an instance randomly chosen . The area where the two algorithms diverge is on the way the committee of hypotheses is created. Where QBC uses the randomized Gibbs algorithm, Query by Bagging utilizes the Bagging algorithm.

Bagging is a re-sampling algorithm that uses sampling with replacement on the training set in order to create a different set of instances of the same size as the initial training set. Some instances are thus present more than once and some others are not present at all. The sampling with replacement is performed t times where t represents the number of hypotheses that will represent the committee. The bagging algorithm has shown to reduce the variance component of the prediction error [1] as well as improving the

Algorithm 2 Query by Bagging

Input: The following are required:

N : Number of iterations

L : Classification algorithm

T : Number of time Bagging re-samples the data

S : Number of randomly picked unlabeled instances

U : Set of unlabeled data

Initialization: Small training data $D_1 = (x_1, y_1)$

1. **for** $i = 1, 2, \ldots N$ **do**
2. Sample with replacement t times from the training set according to uniform distribution and produce subsets $c_1, c_2, \ldots, c_T$
3. Run L using each subset c_j as a training set to produce hypotheses $h_1, h_2, \ldots, h_T$
4. Randomly generate a small set of S points from U with respect to uniform distribution over U. We call the set C.
5. Choose an instance $x^* \in C$ such that it brings most disagreement among hypotheses
   $x^* = arg\ min_x |\{t \leq T \mid h_t(x) = 0\} - \{t \leq T \mid h_t(x) = 1\}|$
6. Provide the instance x^* to a domain expert for labeling and obtain the label y^*
7. Append the newly labeled instance (x^*, y^*) to the training set:
   $D_{i+1} = D_i + \{(x^*, y^*)\}$
8. **end for**

Output: The final hypothesis is as follows:

$h_{fin}(x) = arg\ max_y |\{t \leq T \mid h_t(x) = y\}$

where h_t are hypotheses obtained during the last iteration.

prediction accuracy on future data.

Algorithm 2 illustrates the way Query by Bagging performs Active Learning. Query by Bagging relies on the Bagging algorithm to resample the data a number of times to obtain the different hypotheses. A small subset of the unlabeled data is created and the class label of each instance in the set is predicted by all the hypotheses. The most useful instance is selected from those newly labeled instances, based on the predictions by the hypotheses by using the notion of margins. The margin is defined as the difference between the number of votes by a committee for the one class label against the other class label in a binary classification, or between the number of votes by a committee for the most popular class label and the second most popular class label for a multiple classes classification [56]. The instance with least margin is appended to the existing training data and the whole process is repeated again.The final hypothesis is therefore obtained by majority vote using the hypotheses in the final iteration or by averaging the output from the hypotheses obtained in the last iteration.

4.4 Query By Boosting

Query by Boosting was introduced at the same time as Query by Bagging by Abe and Mamitsuka in their article "Query Learning Strategies using Boosting and Bagging" [1]. This Active Learning algorithm follows the general concept of Active Learning in the sense that it chooses the next query point as the point that creates most disagreement among obtained hypotheses. Query by Boosting employs a boosting algorithm "Adaboost", described in Section 3.2 of Chapter 3, to create its committee of hypotheses. Adaboost works by increasing the weight of the instances incorrectly classified at each iteration, hence forcing the classifier to focus on those particular ones. Therefore, Adaboost is frequently used to boost the performance of the underlying classifier.

Since the original Query by Committee is computationally expensive due to the use of the randomized Gibbs algorithms, Query by Boosting combines the main idea of the Query by Committee approach which uses selective sampling using the margin and the re-sampling boosting algorithm which has proved to increase the performance of the underlying learner. Algorithm 3 takes us through the steps required to perform Active Learning using the Query by Boosting (For detailed information about the Adaboost algorithm, please see Chapter 3 Subsection 3.2.2).

Algorithm 3 Query by Boosting

Input: The following are required:

N : Number of iterations

L : Classification algorithm

T : Number of time Adaboost re-samples the data

S : Number of randomly picked unlabeled instances

U : Set of unlabeled data

Initialization: Small training data $D_1 = (x_1, y_1)$

1. **for** $i = 1, 2, \ldots N$ **do**
2. Run Adaboost using as input: the training data D_i, the classifier L and the number re-sampling is done T. Adaboost produces: $h_{fin} = arg\ max_y \sum_{h_t(x)=y} log\frac{1}{\beta_t}$
3. Randomly generate a small set of S points from U with respect to uniform distribution over U. We call the set C.
4. Choose an instance $x^* \in C$ such the margin is the smallest as follows $x^* = arg\ min_x |\sum_{h_t(x)=0} log\frac{1}{\beta_t} - \sum_{h_t(x)=1} log\frac{1}{\beta_t}|$
5. Provide the instance x^* to a domain expert for labeling and obtain the label y^*
6. Append the newly labeled instance (x^*, y^*) to the training set: $D_{i+1} = D_i + \{(x^*, y^*)\}$
7. **end for**

Output: The final hypothesis is the h_{fin} obtained in the last iteration.

4.5 Additional Remarks

The three algorithms mentioned above Query by Committee (QBC), Query by Bagging (QBag) and Query by Boosting (QBoost) are commonly used when performing Active Learning. QBC uses one version of the training data and is only limited to using random algorithms (such as the Gibbs algorithm) that can generate different hypotheses for the same training data [1]. On the other hand, QBag and QBoost employ different versions of the training data generated by using the bagging and boosting re-sampling algorithms respectively.

Abe and Mitsuka demonstrated that QBag and QBoost achieve a better performance than the QBC [1]. The good performance of the underlying bagging and boosting sampling algorithms reflects on the good performance of the QBag and QBoost. Both bagging and boosting seem to significantly reduce the predictive error. It has been shown that bagging and boosting both reduce the variance component of the prediction error while boosting also attempts to reduce the bias component of the error [11, 33, 3]. However, in order for QBag and QBoost to perform well, good ensembles need to be created and this requires that the learning algorithm be *unstable*, which means that the output model should be sensitive to any small changes in the training data [27]. Further, Dietterich showed that boosting has the tendency to overfit in presence of a high level of noise [27]. In fact, boosting will continuously increase the weight of mislabeled examples (in this case outliers) which will lead to overfitting. In contrast, bagging stays robust when the training data contains a significant amount of noise.

4.6 Summary

This chapter focused on "Active Learning" and how ensemble learning is used to perform Active Learning. In the real world, where training data is known to be hard to find or hard to generate, Active Learning has proved to be a strong technique to tackle that issue. Active Learning allows us to iteratively choose those instances that, when added to the training data, will bring most information. As seen in this chapter, Active Learning calculates the degree of uncertainty with regard to predicting the class label of instances. The uncertainty is the highest when the committee of hypotheses disagree the most. Choosing instances that generate most uncertainty in the committee and adding them to the training dataset will cause the number of training data needed to create a good model to decrease. Therefore, a smaller set of labeled examples generated through Active Learning, will be able to predict the class label of unseen data with the same or better prediction accuracy than a bigger training set randomly created from the unlabeled set.

Part II

LEARNING THE DE-DUPLICATION FUNCTION

Chapter 5

Experimental Design

Recall that the purpose of the de-duplication function is to be able to determine those records in the database that refer to the same real-world object, but that are not syntactically identical due to the presence of typographical errors, misspellings or abbreviations [6]. The detection of duplicate records is especially crucial in the domain of record linkage. In fact, in this case, one needs the determine if two records, from different database sources, refer to the same entity in the absence of a common identification key. It is also crucial in the domain of data integration where two different database sources might have tables containing the same information. The task of identifying two records as duplicates may be made difficult by the existence of semantical incompatibilities. Such incompatibilities include the existence of typographical mistakes in the database, the use of abbreviations or missing data. It can also include the use of synonyms where different expressions can be used to identify the same entity, the use of homonyms where the same identification is employed in two different databases for two different entities [14]. In such situation, a good de-duplicate function is of great importance.

This chapter highlights the structure of our experiment that creates an efficient de-duplication algorithm. The latter makes use of Active Learning performed using ensemble learning. The choice of the ensemble learning methods plays a great role in determining how well the Active Learning algorithm will perform. The ensemble learning method used should be able to create distinct ensembles from the provided dataset. At the same time, each created ensemble should remain consistent with the original dataset with regard to the class distribution and underlying structure of the dataset. In our experiment we contrast two ensemble learning methods namely the "Query by Bagging" and the "Query by Boosting". A thorough description of the de-duplication algorithm is provided in this chapter as well as an explanation of the experimental environment setup.

5.1 Overview of the De-Duplication Method

The De-duplication algorithm we use in this experiment comprises a series of steps that range from data preparation to using ensemble learning methods. Those steps can be aggregated into two phases namely the "Record Mapping" phase and the actual "Active Learning" phase (See Figure 5.1).

In the record mapping phase, similarity functions are used on pairs of records from the original dataset to generate both the training dataset and the unlabeled dataset. Let us bear in mind that each record in a given dataset represents an entity and this entity is defined by a number of attribute values that identify it. Two records representing the same entity will most likely share a certain number of common words. Even though duplicate records might have different or slightly different values for certain attributes, they will most likely have some other attribute values which will generally be identical. Hence, computing the similarity distances between the corresponding attributes of the duplicate pairs (and non-duplicate pairs) will play an important role during the de-duplication process. The resulting distances are passed on to a classifier that will attempt to discern a certain pattern which will guide us when identifying potential duplicate pairs.

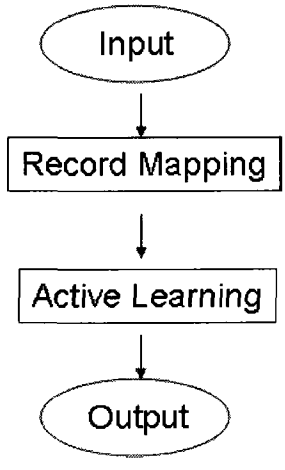


Figure 5.1: Steps of the De-duplication Function

The next phase is the active learning phase. It is the core of the de-duplication algorithm. It actively trains the de-duplication function until the domain expert is satisfied of the result. Using Active Learning allows us to start with a small set of labeled training dataset versus a large pool of unlabeled dataset. Active Learning will gradually increase the size of the training dataset by repeatedly appending new labeled instances, selectively picked from the unlabeled dataset and which contain most information gain. The

benefit of using of Active Learning is that, in the end, the number of training instances needed to produce a good de-duplication model will be exponentially decreased.

The subsequent sections cover the two phases in details and explain the steps involved in each one of them.

5.1.1 Records Mapping using Similarity Functions

Initially, we have two sets of instances: a small dataset T that will be used to generate the mapped training set of approximately 10 to 20 instances and a larger pool of instances U that will be used to create the mapped unlabeled set. The training dataset initially contains manually picked examples of duplicate records and non-duplicates records that will be labeled later. It follows that the size of T is domain dependent. Figure 5.2 illustrates how the mapping is conducted as we will see in the following sections.

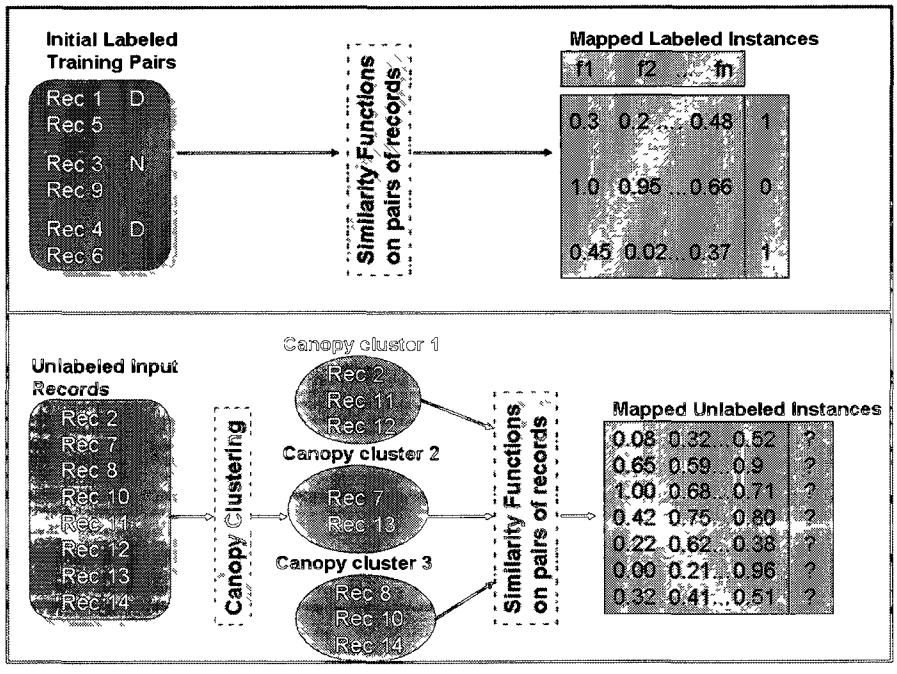


Figure 5.2: Step 1 - Mapping using Similarity Functions

Step 1: The Choice of Relevant Attributes

Our algorithm starts by selecting the attributes that will be the most relevant to our de-duplication task. The choice of the attributes takes into consideration only the attributes that will significantly contribute to the prediction of the class label and therefore removes those attributes that are irrelevant or redundant with regard to predicting the class label. The removal of the impertinent attributes should not bring any noticeable change with respect to data classification. This technique was explained in Chapter 3 and it is referred to as *Attribute Selection*. Its main purpose is to select the smallest subset of attributes such that the resulting class distribution remains as close as possible to the original distribution. Working with fewer attributes without losing any vital information should improve the computation time of the algorithm since fewer attributes are processed [23].

Step 2: Mapping of Training Dataset using Distance Functions

The next step consists of mapping the records in the small dataset T into new records having similarity distances as their attribute values (Figure 5.2 illustrates this mapping). We refer to this new dataset as T_p. The creation of the mapped training data T_p starts by choosing a set of similarity functions in accordance to the types of attribute values present in the set T (numbers, abbreviations, short strings, long strings etc). For instance, the "Jaro" similarity function would be a good choice if the values of an attribute were shorter strings, or the "Soundex" if it is believed an attribute might have phonetic misspellings such as the misspelling of the first name *Kaitlyn* into *Kaitlin* or *Kathlynn*. Therefore, for each pair of records (duplicate or not) present in the set T, the similarity distance is computed between the corresponding attribute values, producing new records containing the distance information of the pair. At the end of each new record, we append the class label indicating whether the pair in question is a duplicate pair or not. The values of the class label can either be '1' for duplicates or '0' for non-duplicate.

The resulting records, which makes the mapped training set T_p, may have the same number of attributes as the original instances in the set T, or have more attributes. This will depend on whether one similarity function was used per attribute or if multiple similarity functions were applied to the same attribute which would produce multiple fields for that one attribute. Given the existence of a large number of distance functions, sometimes many functions may be appropriate for a particular attribute. For instance, we might choose to use the Soundex similarity function and the Levenshtein edit distance on the same attribute. Soundex would determine how phonetically close the two records are for this particular attribute. The Levenshtein edit distance would detect how close the strings are when we take into consideration the position of their corresponding characters.

Another example would be to use a token-based similarity function as well as a distance-based function. Using multiple similarity functions for the same field help gather more information on how close the two values are to each other. It follows that we have to be careful not to utilize similarity functions that provide the same kind of information on the same attribute.

Here is an example of a duplicate pair mapped:

Robert, Smith, 44, 32 street, ottawa, k2b5p3
Bob , Smith, 44, 32 st. , ott. ,k2b5p3
⇓
0.67, 0.78, 1.0,1.0, 1.0, 0.79,0.82, 1.0, $\lfloor \mathbf{1} \rceil$

where the last digit of the resulting record indicates that the original two records represent the same entity (here the same person).

Step 3: Mapping of the Original Dataset

The mapping of the records in the dataset U is done in a very similar way to the mapping of the training data explained above. Pairs of instances are chosen from a large pool of data U (the original dataset from which duplicates are to be searched) and similarity functions are used on the pairs in order to compute the affinity that exists between them. This will produce new records with distance values as their attribute. The only difference between these mapped records and the mapped training records from the previous step, is that they do not have a class label identifying them as identical or not (Figure 5.2 illustrates this). Hence the name of unlabeled data. These new records will have to be labeled at a later time by the trained de-duplication model.

Given that duplicates will have to be identified from the unlabeled pool of records with distance values as attribute. it is imperative that all possible duplicate pairs in the original dataset be coupled and compared using similarity functions. To capture all the duplicate would require us to perform a Cartesian product of the set U with itself: U x U. The generation of all possible pairs in the set U is recommended when the size of the set is very small. However, as the number of instances in the set U increases, the number of all possible instance-pairs increases exponentially. making it difficult to manage. Alternative solutions of detecting potential matches are therefore needed.

Our De-duplication function avoids using the Cartesian product while drawing the pairs from the set U for mapping. Instead, it relies on the use of a blocking methods. as explained in Section 2.3. A good blocking method will strategically cluster all potential duplicate records into the same group, using simple techniques that do not require much

computations It is common that few false matches (non-duplicates records) be added into those groups. In the end, a record with all its duplicates should be found in the same blocking group.

After the groups are created we proceed with the mapping in the same way as for the training set, with the only distinction that the resulting records of distance values will have no class label specified. Comparatively, the mapping of the data in the set U is done on the attributes that were selected as the most relevant attributes during the attribute selection Note also that the same similarity functions that were used on the small set T are used on the set U. For instance, if the "Soundex" and the "Smith-Waterman" similarity functions are used on the first attribute of the set T, the same functions will be used on the same attribute in the set U The result of this mapping will be a new set U_p of distance-value records that have the same number of attributes as the mapped records T_p but with equal or more attributes than the original records in U.

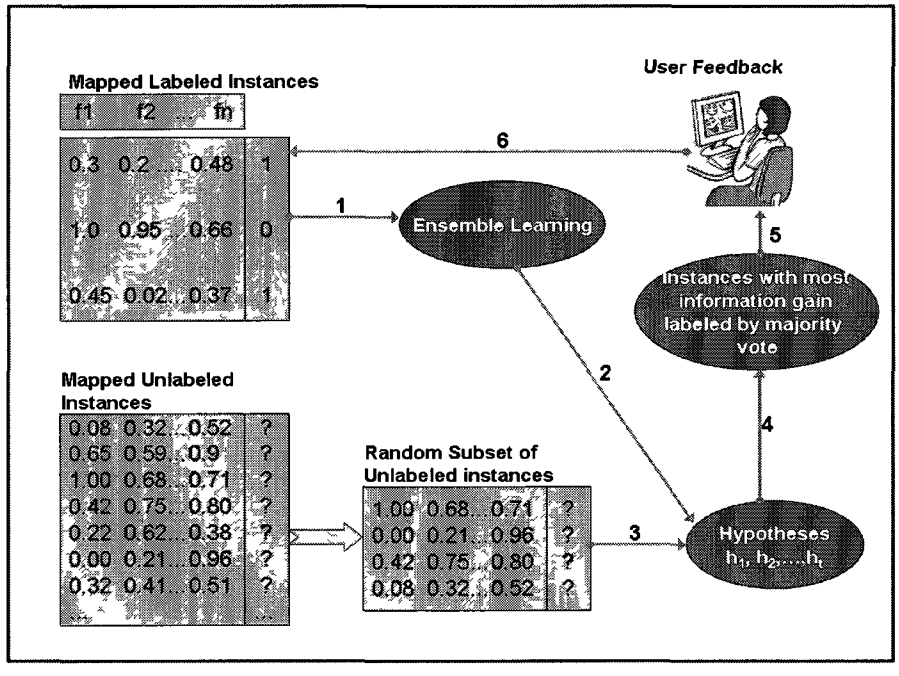


Figure 5.3 Step 2 - Active Learning using Ensembles

5.1.2 Active Learning for the De-duplication Function

At this point of our algorithm, we dispose of two sets of data, namely the training dataset T_p and the unlabeled dataset U_p. The set T_p will be used to train a learner to detect duplicates from the set of unlabeled instances U_p. To achieve this, we take advantage of "Active Learning" described earlier in the Chapter 4. The active learning process used is illustrated in Figure 5.3.

Description of Active Learning

Figure 5.3 illustrates six important steps used by our de-duplication function when it is actively training the learner to produce a better predictive model. This model will later be used to predict the class label. The six steps of Active Learning are continuously repeated until a satisfying degree of accuracy and precision is reached.

Steps "1" and "2"

First, the training set is taken as input and fed to an ensemble learning method. The latter works by first forming different subsets of the training data using a resampling formula. The chosen resampling method will create subsets very similar to the original training set but different among each other. A learning algorithm (such as a decision tree) is then trained on each subset. As a result, an ensemble (or committee) of slightly different classifiers is generated. Each classifier represents a predictive model.

The generated predictive models will then be used (in the next step) to predict class label of certain unlabeled records from U_p and the predictions will be compared to detect any disagreement among models.

Steps "3" and "4"

The next phase consists of choosing, amongst the large pool of unlabeled instances, those that are considered to be the most informative in the sense that when they are appended to the training set, the learning process of the classifier significantly improves, thus producing a better predictive model. Here is how the selection process is conducted.

We begin by first creating a subset u_p of the large pool of unlabeled instances U_p by using a simple random sampling from the set U_p. Each instance in the new set u_p is fed to the committee of classifiers previously created by ensemble learning and each classifier predict the class label of the given instance. A majority vote for the class label is performed. If the result shows that the difference (the margin) between the number of votes for the "duplicate" class label and that for the "non-duplicate" class label is very

small, it means, if the predictive models disagree on the class label. then that particular instance is marked as an informative instance. A tentative class label, obtained from majority voting from the committee, is appended to the instance for later review by the user. Generally, the number of most informative instances passed to the user for validation in each iteration can be determined beforehand. For instance, we can choose that only one record causing most disagreement be passed to the user at each iteration or a set of, let's say, five records with most disagreement. Again, it follows that this is domain dependent.

Note that the reason behind choosing a smaller set to represent U_p is to reduce the high cost that would result from having to perform the prediction of the class label on the entire pool U_p on each iteration.

Steps "5" and "6"

In the last two steps, the instances marked as containing most information with regard to class prediction are presented to a domain expert (a knowledgeable external user). The latter verifies the suggestion given by the committee of hypotheses and validates it, or rectifies it, accordingly. The properly labeled instances are thereafter added to the training set.

The entire learning process is repeated, each time using a larger training set than the previous iterations. This will cause the ensemble learning method to create new hypotheses that will be different from the hypotheses generated in the previous iterations. As we continue, we can expect a decrease of the number of instances that cause disagreement among hypotheses and an improvement of the accuracy of the predicting model.

To investigate the stopping criteria, the user repeatedly check at different points the performance of the training output. To to this, the user trains the learner using the generated training set and uses the ten-fold cross validation technique to estimate how well the created model will perform on future unlabeled data. If the user is satisfied with the performance of the model, then the process may be stopped and the model created will be our de-duplication function. However, if the user is not satisfied, the process is repeated until the domain expert is satisfied with the createdthe created training set generates satisfactory performances.

5.2 Environment Setup

This section covers the methods and techniques used in each step of the de-duplication process. Our experimental design takes advantage of a number of pre-existing data mining techniques. Since various machine learning algorithms are used throughout our active learning session, the preparation and evaluation of the de-duplication function is mainly performed using WEKA. a software that has a collection of many machine learning algorithm ranging from classifiers to attribute selection methods [80].

5.2.1 The Datasets

The datasets used for the evaluation of the de-duplication function are datasets that are generally used by other researchers in different communities whose study focuses mainly on *duplicate detection, record linkage* or *field matching* [50, 5, 66]. Duplicate records in these selected datasets are already identified. They are put in the same block and are assigned the same block number. Table 5.1 gives a summary of the datasets.

The first dataset used in the evaluation is the "RESTAURANT" dataset. It contains a collection of 864 records, 533 of which is drawn from Fodor's restaurant guidebook and the remaining 331 from Zagat's restaurant guidebook. The data in this dataset is labeled and contains a total of 112 duplicate records. The attributes used are the restaurants name, address, city and the type of cuisine. This dataset was downloaded from RIDDLE (Repository of Information on Duplicate Detection, Record Linkage and Identity Uncertainty) which is publicly accessible and made available by the department of Computer Science at the University of Austin.

The next dataset used is the "CENSUS" dataset. The records in the set. as the name of the dataset suggests, contain census data which is mostly information collected about each member of a household in a particular geographic region. The attributes used to describe members of a population are the last name, first name, middle name, house number and the street address. This dataset contains 841 records which were created by merging two duplicate-free datasets of 449 and 375 records each. The merging introduced 327 duplicates.

Four different synthetic mailing list datasets were also generated using the CUCS mailing list data generator created by Mauricio Hernández. This data generator allows us to control how the datasets are produced, how many records are created, how many duplicates one record can have, and the amount of error to be introduced in the duplicated records in any of the attribute fields. The default values for the different parameters are based on real world distribution and are known frequencies from studies in spelling

Datasets	Instances	Duplicated Recs	Max Dups	Missing Values
Restaurants	865	112	2	No
Census	841	327	2	Yes
Mailing 1k	996	339	3	Yes
Mailing 1k(extra errors)	1016	339	3	Yes
Mailing 2k	2022	672	3	Yes
Mailing 5k	4949	1630	3	Yes

Table 5.1: Summary of Datasets used

correction algorithms [39].

For this particular experiment, three datasets were created by changing the size parameter. Hence, datasets of $\approx$ 1000 records (996 records), $\approx$ 2000 records (2022 records) and $\approx$ 5000 records (4949 records) were generated. Note that the generator does not provide the exact number of records requested but rather an approximate number. For all the datasets, we set the parameters so that we have an average of two duplicated records. All the other parameters controlling the amount of error introduced in the dataset are kept as default. The last dataset was created in the similar way as the previous dataset with a data size of $\approx$ 1000 records (1016 records) but with more errors inserted. The default values were calibrated to simulate real-world common typographical errors in datasets.

5.2.2 The Experimental Setup

- As described earlier, the first step before we start the mapping of the datasets is to perform an "Attribute Selection" in order to only work with relevant attributes. To perform this task, we used the *Wrapper Subset Evaluation* in WEKA [79]. As mentioned in Chapter 3. the wrapper approach provides better results that are best fitted for the learning algorithm than the filter approach. The wrapper approach performs a search of a good attribute subset by using the learning algorithm itself as part of the function evaluating feature subsets while the filter approach does not use an underlying learning algorithm [46]. The wrapper subset evaluation method analyzes subsets of attributes with regards to their usefulness to the class label. We used a 5-fold cross-validation to estimate the accuracy of our learning algorithm which is the C4.5 decision tree (or J48 in WEKA). The search for a good subset is done by searching the space of all possible feature subsets. To represent the different subsets, each attribute is given a bit to represent the state it is in, "0"

indicating that the attribute is absent from the subset and "1" indicating that the attribute is present. The goal then is to choose the optimal subset such that the generalization error of the learning model is the least. However, if the number of attributes is large. the exhaustive search becomes very expensive or impossible. Therefore, instead of an exhaustive search, alternative methods of search are used in order to evaluate only a small number of variables. In our experiment, we used the *Rank Search algorithm* with the *Information Gain Attribute Evaluatation.* The way it works is by ranking the attributes according to their information gain. Then, the error rate of the increasing set of attributes is computed. starting with the most informative attribute, then the most informative attribute with the second most informative attribute and so on.

- The mapping of the training dataset is the next step. The mapping involves finding distance values for each pair of records provided in the small dataset T to form a new set which will be the training set. We decided to used one similarity function per attribute. Since our datasets are mainly records of text data, we opted to use on all the attributes the *Smith-Waterman-Gotoh* similarity function described in Chapter 2. This function was proved to be computationally inexpensive [36]. It finds the minimum cost of aligning two strings in only $O(m, n)$ computational time where m and n are the lengths of the two strings. Instead of finding all of the optimal alignments, the algorithm only attempts to find just one of the optimal alignments [36] and it is therefore suitable for our task.

- Before the mapping of the large pool of instances to label U. which consists of applying the similarity functions on the set. we first use a blocking method to produce smaller subsets of the set U. We thus avoid the cost of having to pair up and apply the similarity functions on all the instances in the set U. Instances in the same subsets will be similar to each other (with regard to distance values) but different to other instances in different subsets. During our experiment, we evaluated the de-duplication algorithm with two different blocking methods.

  We first carried out the blocking task using *Canopy Clustering* (Section 2.3). Canopy Clustering produces overlapping clusters referred to as **canopies**. For our evaluation, we implemented this algorithm, in Java, and added it into WEKA. When Canopy Clustering is measuring the distances between the instances, it requires two threshold distance values that will be used to group close instances in the same canopy. We used a separate but similar validation dataset as the dataset at hand in order to tune those two values. In addition, our canopy clustering uses

the "Tf-Idf" similarity function.

We ran the same experiments, but this time using *Bi-gram Indexing.* Bigram Indexing, as explained earlier in Chapter 2, is another efficient blocking method that first creates bi-gram lists from blocking key values, then uses a specified threshold value to determine how many bi-grams will be combined in order to create sublists of all possible combinations. The latter are put in an inverted index which will be used to identify records in a block. For this specific experiment, we used a threshold value of 0.8, which we set by inspection.

- Once the blocks are created, the *Smith-Waterman-Gotoh* similarity function is used to map the large set "U" into the unlabeled set "U_p". At this point in time, we are in possession of two datasets that are at the heart of the active learning process: the training dataset "T_p" and the unlabeled dataset "U_p" (Section 5.1.1). In our experiment, Active Learning is accomplished using Query by Bagging (QBag) and Query by Boosting (QBoost). WEKA already included the Bagging and Adaboost.M1 algorithms. Therefore, we incorporated Java code into WEKA to emulate Active Learning with QBag and QBoost. Note also that the two algorithms were used in conjunction with the decision tree algorithm C4.5. The reason for the choice of these two learning algorithms as our principle methods of Active Learning is as follows. QBag and QBoost create subsets of the original dataset that are diverse but yet remain consistent with the original dataset. Thus, when a weak classifier is applied to each subset, the resulting ensembles will generate different hypotheses that can be polled through majority vote when performing class prediction. The diversity of the hypotheses ensures better results when searching for the disagreements among hypotheses [56].

5.3 Summary

In this chapter, the details of the proposed algorithm for mining duplicates from a large dataset was provided. As described above, the de-duplication function is divided into two major steps. The first step maps a small set of duplicates and non-duplicates examples into the training set and maps the large pool of data instances to label into the unlabeled set. The second step, Active Learning, is used to teach a learner to produce a good model that will efficiently predict whether a pair of unlabeled instances is duplicate, or not. In addition, this chapter motivates the choice of methods and techniques used throughout the de-duplication process. The results from the evaluation of the proposed de-duplication function are highlighted in the next chapter.

Chapter 6

Evaluation and Results

The strength of the de-duplication method presented in Chapter 5 is the ability to actively teach a learner how to distinguish duplicate and non-duplicate records. When performing Active Learning, ensemble learning methods are used to create a set of hypotheses that generate slightly different predictive models. These models are given unlabeled records and, for each record, the models predict the class label. The instances that cause most disagreement among the models will be considered as informative and therefore will be appended to the training dataset.

This chapter provides the results of a comparative study performed on the de-duplication function. It also show the different types of evaluation methods used to assess the effectiveness of the algorithm under different circumstances.

Our experiment evaluates the performance of the de-duplication function when using Query by Bagging and Query by Boosting, as introduced in Chapter 4. Furthermore, different parameters of the de-duplication function are modified in order to analyze whether or not such changes have any effect on the effectiveness of the function. We evaluate the function on datasets of different sizes as well as on real world datasets versus synthetic datasets. We evaluate the function on datasets having duplicates with very minor differences and datasets having duplicates with noticeable differences such as very distinct values for the same attribute. We furthermore employ different blocking methods and compare the results obtained.

As shown in Chapter 4, Active Learning does not need a large training set at the beginning of the training process. Indeed, it starts by training the learner with a very small training set. Then, it selectively provides the learner with an augmented training set, now containing additional new and labeled pairs of records that are the most difficult to label. At each step of evaluation, we examine how the usage of Active Learning gradually improves the performance of the trained hypotheses as the training set gets

augmented with new informative records.

6.1 Criteria of Evaluation

The task of evaluating the performance of "data de-duplication using Active Learning" is, in essence, the task of evaluating the performance of a machine learning algorithm. As a matter of fact, our final de-duplication function, which is used to distinguish between duplicates and non-duplicates, is indeed a predictive model. In our case, we use a decision tree. Therefore, techniques generally used to evaluate machine learning methods are used here to evaluate the efficiency of our de-duplication function. The different performance measures used to analyze the function are explained in the next section.

6.1.1 Performance Measures

Confusion Metric

When considering the performance of classifiers, we are usually interested in knowing if the model correctly, or incorrectly, predicts classes. A good way of representing and visualizing the output of a classifier is by using a *Confusion Matrix*. A confusion matrix is represented in the form of a table [79]. In our experiment, the de-duplication function (decision tree) will predict if a pair of two records is either a duplicate pair or a non-duplicate pair. The corresponding confusion matrix may be represented as follows:

		Predicted Class	
		Duplicate	Non-Duplicate
Actual Class	Duplicate	TP	FN
	Non-Duplicate	FP	TN

Table 6.1: Confusion Matrix

Each prediction by the de-duplication function has four possible outcomes. The *true positive* and *true negative* correspond to when the function correctly classifies a pair as duplicates, or non-duplicates, respectively. The *false positive* is when a pair of records is incorrectly predicted as duplicates and the *false negative* occurs when a duplicate pair is incorrectly classified as a non-duplicate pair.

Using the confusion matrix, we evaluate the performance of the classifier by utilizing performance measures that are based on the output of confusion matrix. The following sections highlight the performance measures used in our experiment.

Accuracy

The accuracy is defined as the overall correctness of the model. It is the number of correctly classified examples divided by the total number of examples. Using our confusion matrix, Accuracy is represented by the following formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

It has been shown, however, that this measure is not always a reliable way of measuring the performance of a classifier [62]. As a matter of fact, in an imbalanced environment where two different classes are present in different proportion, this measure may yield misleading results [62].

Precision

In our de-duplication context, Precision is defined as the proportion of true duplicates among all the instances predicted as duplicates. The formula is as follow:

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

Recall

The Recall measure is the proportion of instances which are classified as duplicate records, among all instances which truly are duplicates. The formula, based on the confusion matrix, is a follows:

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

Put in other words, Recall computes the portion of duplicates, from all real duplicates, that was captured by the predictive model.

F-Measure

There is a trade-off between the previous two performance measures, Precision and Recall. As the precision goes up, the recall tends to go down, and vice versa. The F-Measure takes both measures into account and is formulated as follows:

$$\text{F-Measure} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6.4}$$

Note that the F-Measure is the weighted harmonic mean of Precision and Recall and is usually also referred to as F1 Measure because recall and precision are evenly weighted [54]. The general formula is shown below. For the F-Measure, the term β (which indicates the importance of recall relative to precision), is equal to 1.

$$F_\beta = \frac{(1 + \beta^2) \cdot (Precision \cdot Recall)}{(\beta^2 \cdot Precision + Recall)} \tag{6.5}$$

Mean Absolute Error

The Mean Absolute Error (MAE) is defined as a measure used to compute how close predictions are to the eventual outcome. This measure is computed as follows [79]:

$$MAE = \frac{|p_i - a_i| + ... + |p_n - a_n|}{n} \tag{6.6}$$

where p are predicted values and a are actual values.

6.2 Methodology and Experimental Results

For each dataset, before Active Learning is conducted, the dataset is first preprocessed. As mentioned in Chapter 5, this consists of performing an attribute selection on the dataset to only work with attributes that are of interest to our task and then running a blocking method on the dataset. Once the instances are grouped into blocks of potential duplicates, instances in the same block are paired up and similarity functions are run on the pair to create new records now consisting of distance values as their attributes. The unlabeled dataset is then created. Next. few instances are chosen to create the training dataset. By inspection, we chose to start with a training set of 10 instances - 5 duplicates and 5 non-duplicates.

With the training set and the unlabeled set provided. Active Learning is performed and the results are provided in the section below.

As mentioned in Chapter 4, Active Learning using Query by Bagging or Query by Boosting works iteratively by appending new meaningful and informative instances to the training set and then re-training the classifier. In an attempt to examine the informativeness of each instance selected and added to the training set as well as its impact on the performance of the classifier, we only append one instance at a time to the training

set at each iteration. After 10 instances are appended to the training set, the corresponding predictive model is saved and used to evaluate the classifier at that particular point. Training sets of increasing size (the size number being a multiple of ten) and their corresponding models are continuously saved until the performance of the classifier is determined satisfactory. This indicates that the classifier can effectively identify instances as being duplicates or non-duplicates. Each produced and saved model is therefore evaluated against the unlabeled dataset using the WEKA experimenter application software and the different performance measures are computed.

It is expected that the performance of the de-duplication function (or classifier) improves as more and more informative instances are appended to the training set. The evaluation is performed using ten-fold cross validation.

6.2.1 De-duplication using Query by Bagging

We first evaluate the de-duplication function using Active Learning based on Query by Bagging. The performance of the entire process is analyzed on different datasets which are commonly used in the de-duplication community. It is important to note that this experiment was conducted in a way that will allow us to study the Active Learning behavior in different situations. We study the impact of varying dataset size on the performance of the Active Learning process, using real-world datasets versus synthetic dataset and finally changing the blocking method used in the pre-processing step.

Active Learning using Query by Bagging is first evaluated against the different datasets as introduced in Chapter 5. Figures 6.6(a) and 6.1 shows the results of the evaluation on the "Restaurants" and "Census" datasets. The next evaluation is performed on the Mailing dataset. As explained in Chapter 5, we used the mailing dataset generator and produced three datasets of different sizes, namely Mailing1k, Mailing2k and Mailing5k. The results of the evaluation on the three datasets are seen in Figures 6.2 to 6.4. For these three generated datasets, we changed the dataset size parameter and kept the remaining parameters as default. The remaining parameters include the percentage of errors induced for different types of errors used while creating duplicates records. For all the above datasets, we used the Canopy Clustering as the blocking method.

Figure 6.6 compares the performance of Active Learning using Query by Bagging in two different circumstances, first when the Canopy Clustering blocking method is used and second when bi-gram indexing is used.

The results of our experiment are illustrated in these graphs. Each graph shows the values of the computed performance measures on the de-duplication function when the

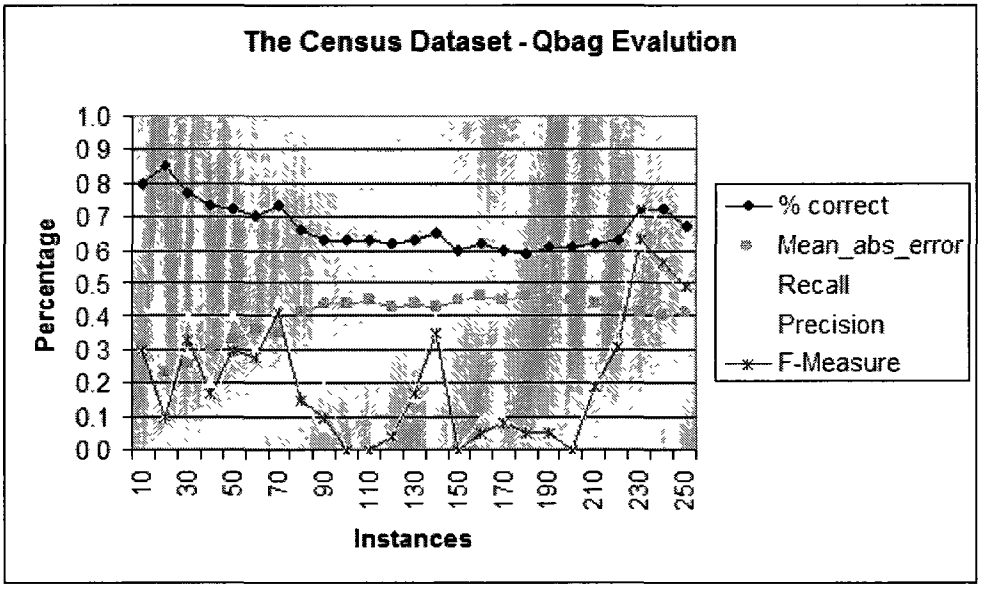


Figure 6.1: Evaluation using Canopy Clustering on the Census Dataset

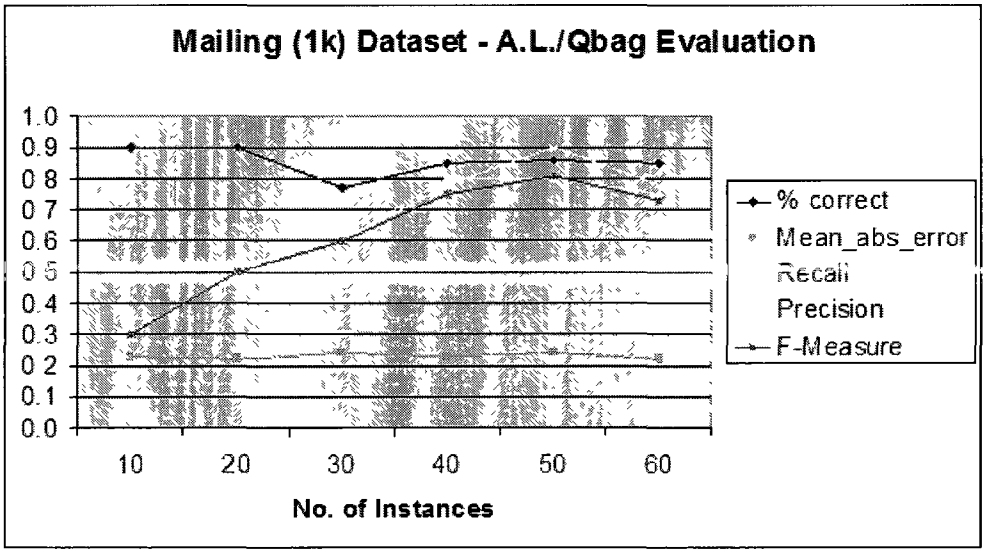


Figure 6.2: Evaluation using Canopy Clustering on the Mailing Dataset ($\approx$ 1000 records)

final function is obtained at different stages of the Active Learning process. As a matter of fact, we take the predictive model generated using the original training set of 10 instances and evaluate its accuracy, recall. precision, F-measure and the absolute mean error. We then start with our Active Learning and get the predictive model generated

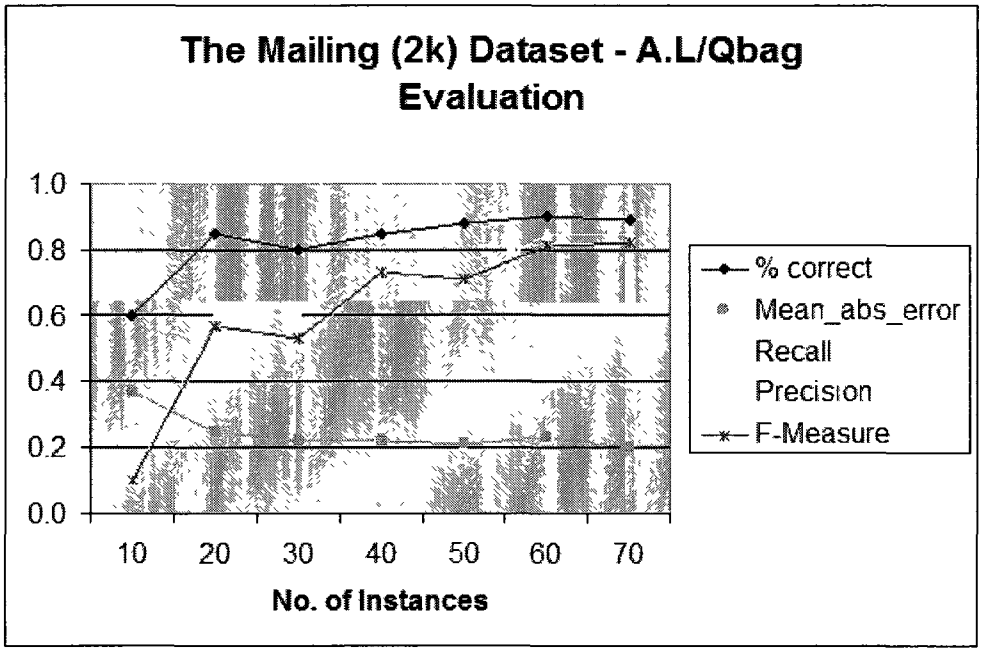


Figure 6.3: Evaluation using Clustering on the Mailing dataset ($\approx$ 2000 records)

with the training set of size 20 and compute the performance measure at that point. We continuously compute the different measures as 10 informative instances are appended to the training dataset. The X-axis on the graph shows the size of the training set when the predictive model is evaluated and the Y-axis the range of computed values of the performance measures.

Discussion

As we observe the resulting graphs, we notice that all the graphs have a somewhat similar behavior. At the beginning of the learning process, the evaluation of the classifier yields poor performance values. However, as the training set receives more and more informative instances, the performance of the generated function progressively improves. This behavior is generally due to the fact that the classifier is repeatedly trained on a better dataset that contains more and more instances. When appended to the training set, these instances bring more information to the classifier. Let us remind ourselves that Query by Bagging first creates diverse ensembles using a resampling technique known as bootstrap sampling. The resulting ensembles slightly differ from each other. The instance that causes disagreement among the ensembles is considered challenging and labeling it with the right class brings tremendous information to the classifier. Therefore,

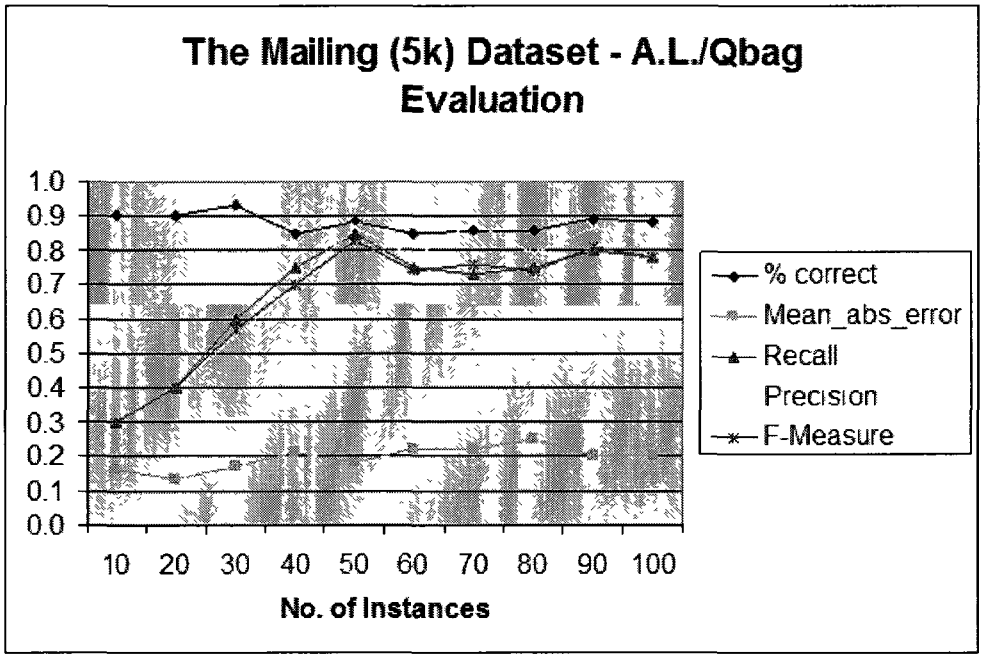


Figure 6.4: Evaluation using Canopy Clustering on the Mailing dataset ($\approx$ 5000 records)

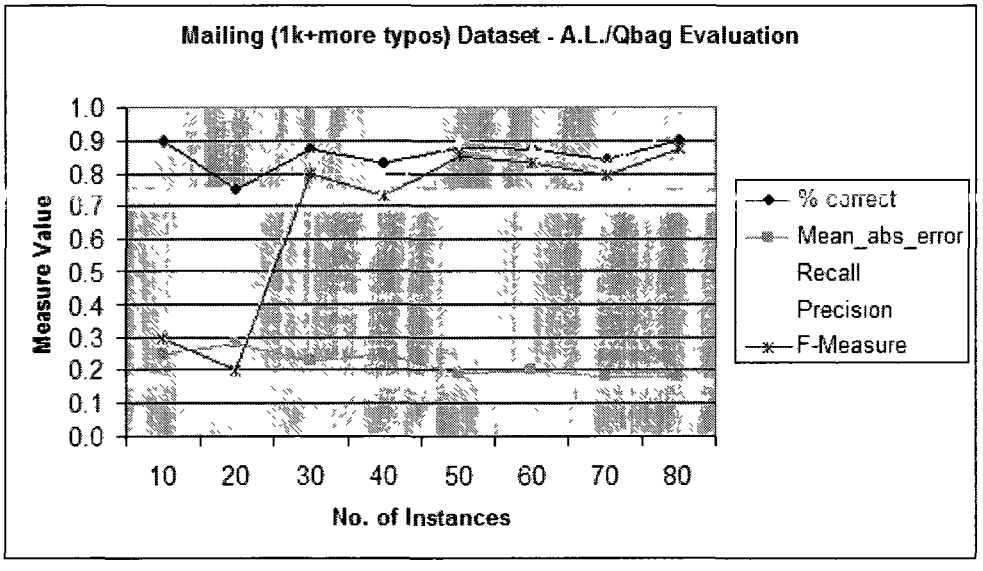
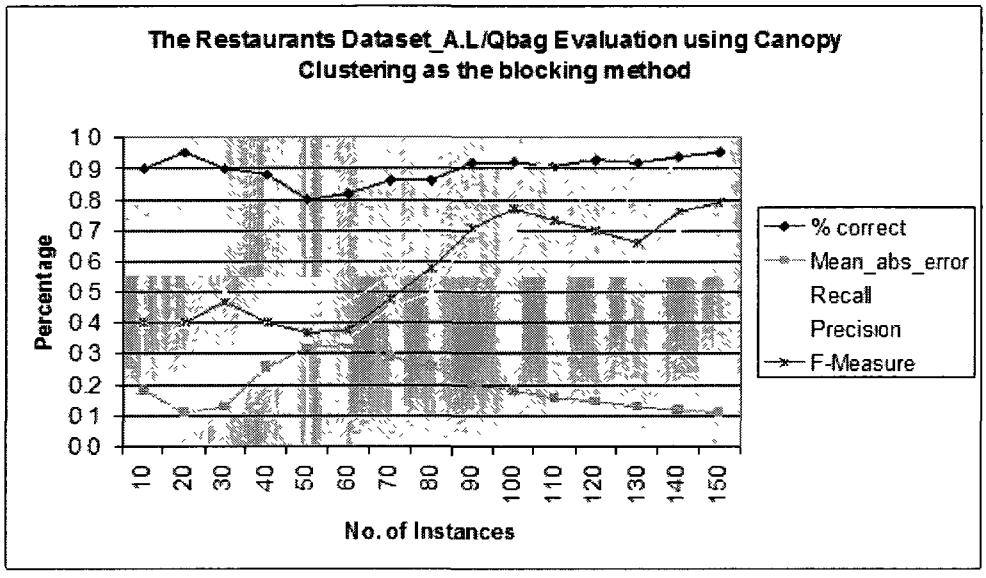
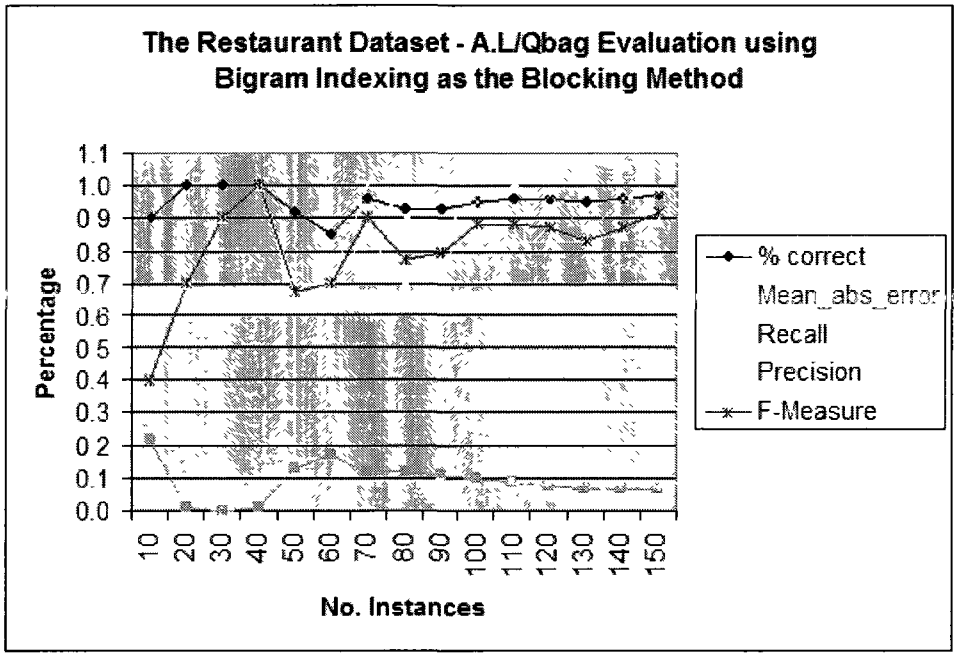


Figure 6.5: Evaluation using Canopy Clustering on the Mailing dataset when the dataset generates more errors($\approx$ 1000 records)



(a) Using Canopy Clustering



(b) Using Bigram Indexing

Figure 6.6: Evaluation using Canopy Clustering when two different blocking methods are used on the same dataset

those disagreement-causing-instances are the reason why, the classifier's performance is improved.

We also examine the performance of Active Learning when the dataset at hand is either a real world dataset or an artificially generated dataset. If we look closely at the evaluation results for the artificial datasets in Figures 6.2 to 6.4, the learning function for all these datasets reaches the best performance quickly when the dataset attains a size of only 50 instances. Note also that the graph shows a steady improvement in performance. Each time 10 instances are added to the training set, the classifier performs better. These three graphs also allow us to study the impact of their size differences. The results generated by the dataset with the smallest size (fig. 6.2) shows a steady and more linear improvement in performance. As the evaluation is done on datasets of bigger sizes, the performance is still good and the learning is just slightly less smooth.

We generated one more dataset using the "Mailing" data generator and we set the probability of inserting an error to a higher value (from the default 25% to 95%). The resulting graph is shown in Figure 6.5. From the graph. we can see that the increase in the duplicate error does not affect the good performance of Active Learning with Query by Bagging.

Now, let us consider the evaluation of the function on real world data: the Census and Restaurant datasets. Active Learning on the "Census" dataset (Figure 6.1) uses a large amount of instances (around 230 instances) to reach a satisfactory performance. During the learning process, some instances that were supposed to be informative actually confused the learner even more. This behavior can be attributed to the duplicates provided in this dataset. In fact, the Census dataset, which contains the identification information of the members of a household such as the name and address, confused the de-duplication process. For example. we have instances such as the following that are indeed true duplicates:

LastName	*FirstName*	*MidInitial*	*HouseNo*	*Street*
PAIGE-WELLS	AGEL	J	231	71ST
PAIGE-WELLS	ADEL		231	71ST

LastName	*FirstName*	*MidInitial*	*HouseNo*	*Street*
SOLLIVAN	BRANDIE	D	14245	22
SOLLIVAN	EVENS		14245	22

and the instances below are in fact non-duplicates. The instances just represent two members of the same household with very close name.

LastName	*FirstName*	*MidInitial*	*HouseNo*	*Street*
RHOADS	JEANNE		110	BRANTWOOD
RHOADS	JANIE	M	110	BRANTWOOD

LastName	*FirstName*	*MidInitial*	*HouseNo*	*Street*
GOINS		L	115	ELDORADO
GOINS		T	115	ELDORADO

If we consider the examples for the entity with the last name "Paige-Wells" and "Rhoads", we can see that both examples are similar. The last names, house number and the street are the same for both instances in each pair; the first names are slightly different and one middle initial is missing. However, these two examples are classified differently. Thus, the classifier is most likely to make random guesses for similar examples, after being trained on these two examples. The evaluation on the Restaurant dataset, Figure 6.6(a), yields better results than the Census dataset. Active Learning reaches a good performance when the size of the training set is around 90 instances. Also, note that the performance lines on the graph are not as smooth as they were for the artificial datasets, which shows that real-world data sometimes contains unpredictable instances that defy the general rule.

The performance of Active Learning using two different blocking methods, the Canopy Clustering and the Bigram Indexing methods, is shown in the graphs provided in Figure 6.6. Bi-gram indexing seems to be a better blocking method. The classifier realizes good performances, early on, and continues to perform well as the size of the training set augments. This means that Bigram Indexing does a better job of grouping together potential match, which then facilitates the learning of the classifier. One possible reason why Bigram Indexing performs better than Canopy Clustering is that Canopy Clustering creates blocks of possible duplicates that are overlapping. After pairing up instances in each block, paired up instances from overlapping clusters will likely be instances with close distances. Therefore, the unlabeled dataset may consist of multiple instances that are similar.

6.2.2 De-duplication using Query by Boosting

The evaluation of Active Learning using the Query by Boosting method is conducted in a very similar way as the evaluation of Active Learning using Query by Bagging described above. We study how well Query by Boosting performs as the Active Learning progressively teaches the classifier with a continuously improved training set. Again,

we analyze the performance of Active Learning using Query by Boosting on datasets of different sizes, on real-world dataset versus artificial dataset and also while two different blocking methods are utilized.

The same datasets used to evaluate the Active Learning using Query by Bagging are used for Query by Boosting. The resulting graphs are also generated in a similar way. The training set continuously increases in size with new instances as the Active Learning proceeds. Each time, after 10 instances are appended to the training set, the performance measures are computed. The graphs show the results of the performance measures computed after the classifier is repeatedly trained using the training dataset that is increased by 10 new instances.

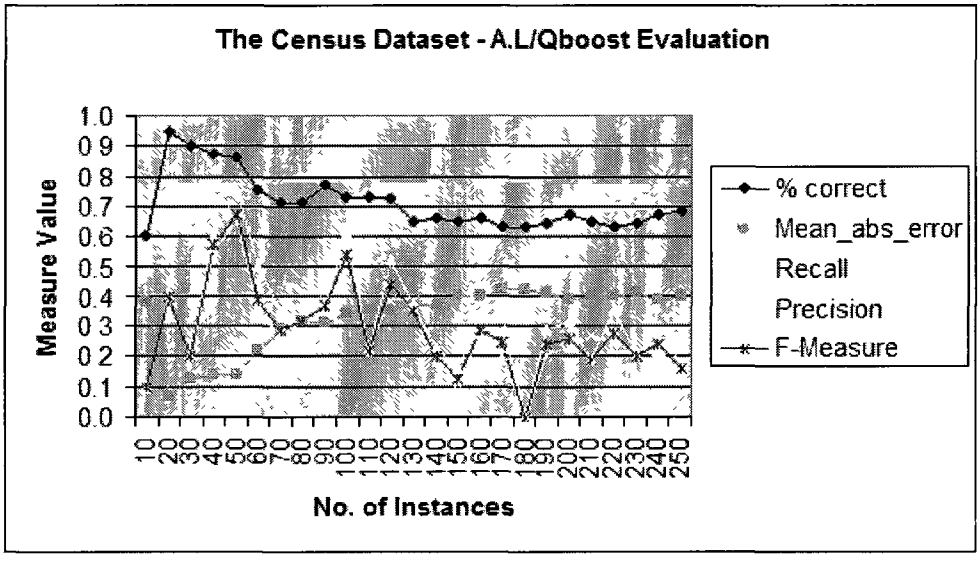


Figure 6.7: Evaluation using Canopy Clustering on the Census dataset

Discussion

As mentioned in Section 4.4, Query by Boosting uses the boosting resampling method to create different ensembles. Boosting works by maintaining a certain weight for each instance in the training set. When creating hypotheses, boosting increases the weight of the instances incorrectly classified by the previous hypothesis, hence causing the classifier to focus on those particular instances in the next learning iteration. In our experiment, Query by Boosting uses "boosting by resampling" where the training set is sampled with replacement with the probability of selection of instances being proportional to their

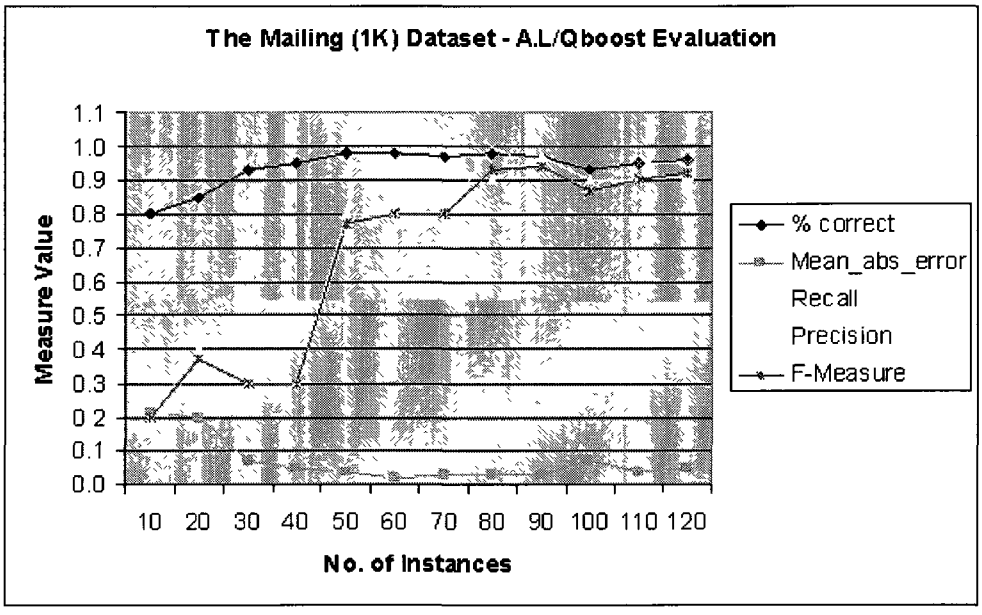


Figure 6.8: Evaluation using Canopy Clustering on the Mailing dataset($\approx$ 1000 records)

weight. The base classifier used is the WEKA version of the C4.5 decision tree called J48.

Figure 6.7 and Figure 6.12(a) show the results of the evaluation of the de-duplication function on the Census and the Restaurants datasets respectively. As the graphs show. Active Learning using Query by Boosting performed poorly on the Census dataset. All the measures used to evaluate the de-duplication function show a decrease in performance as the training set increases. One of the reasons can be attributed to the presence of noise in the dataset. The dataset contains several difficult to learn duplicate examples. Also, when creating hypotheses, at each iteration, boosting measures the error of the current hypothesis. If the error is greater than 0.5, then the process is immediately stopped. This results in fewer hypotheses created. Thus, if for example only three hypotheses are created, the next instance chosen will be the instance that causes most disagreement among the three hypotheses (instead of the usual 10 hypotheses). Hence, we infer that the next instance may not be the most informative.

The evaluation on the Restaurant dataset. as shown in Figure 6.12(a), on the other hand, yields very satisfactory results. Indeed, after only 10 to 20 informative instances are added to the training set, the de-duplication function reached is highly efficient. From that point, newer informative instances bring somehow similar information to classification which causes the classifier to maintain its high performance values throughout.

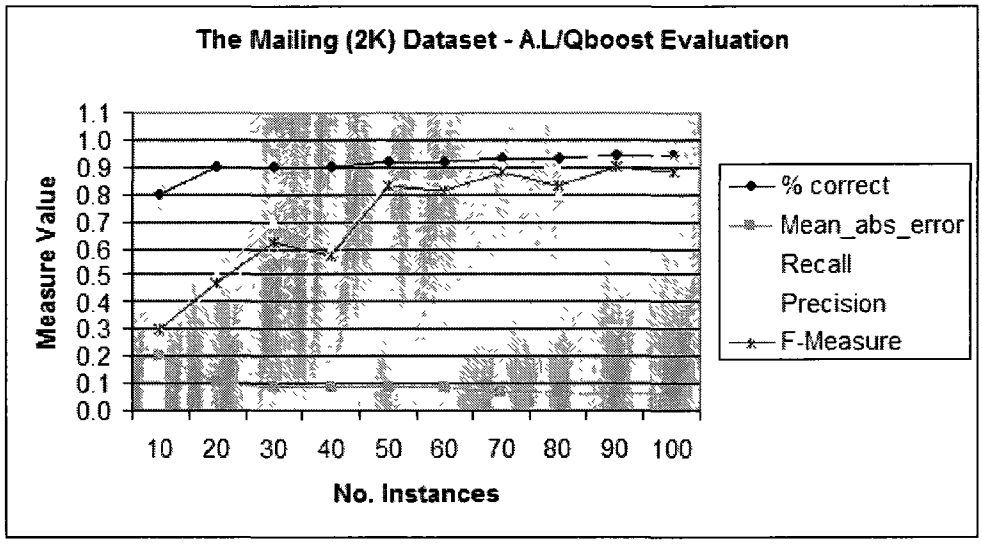


Figure 6.9: Evaluation using Canopy Clustering on the Mailing dataset($\approx$ 2000 records)

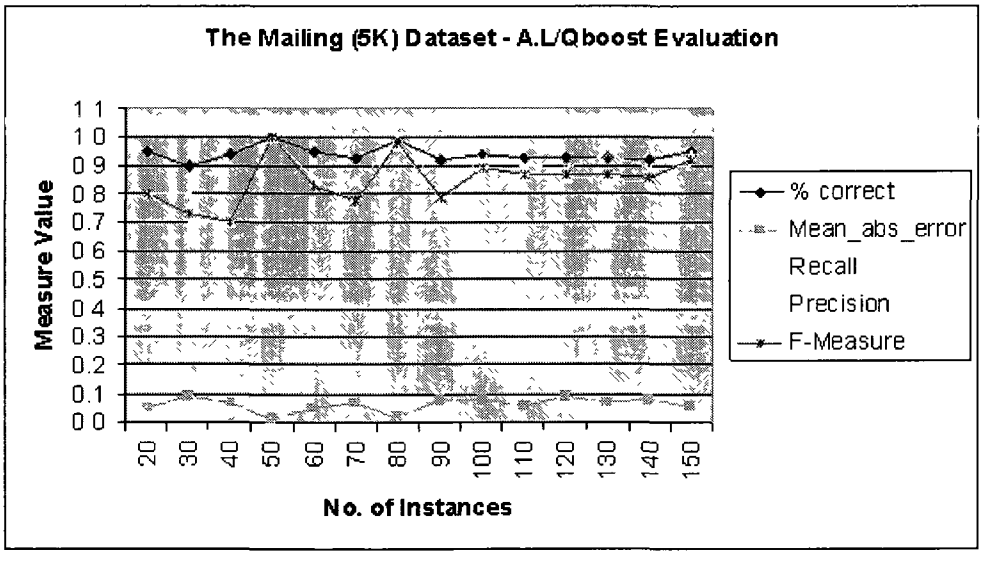


Figure 6.10: Evaluation using Canopy Clustering on the Mailing dataset($\approx$ 5000 records)

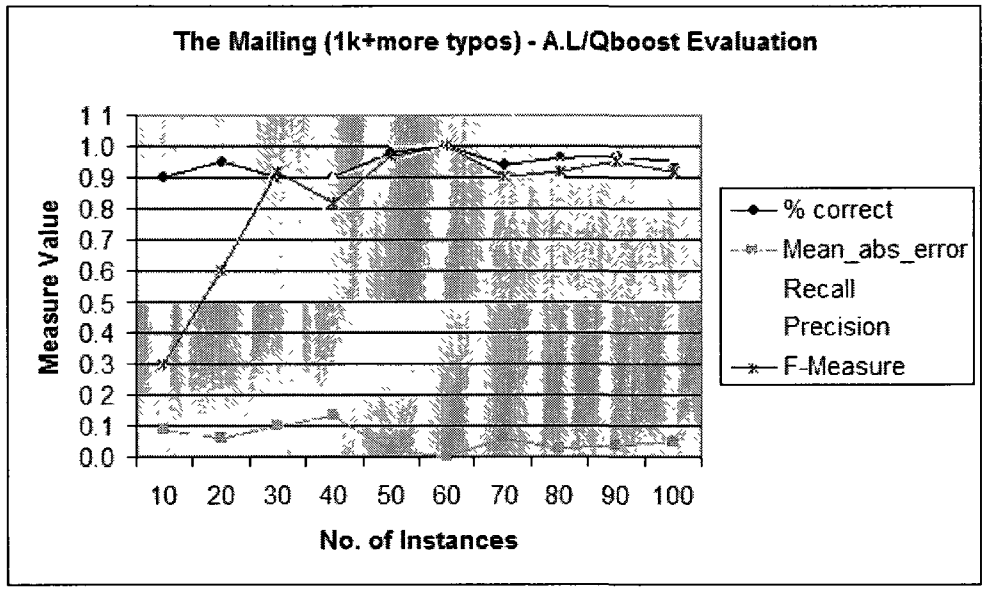
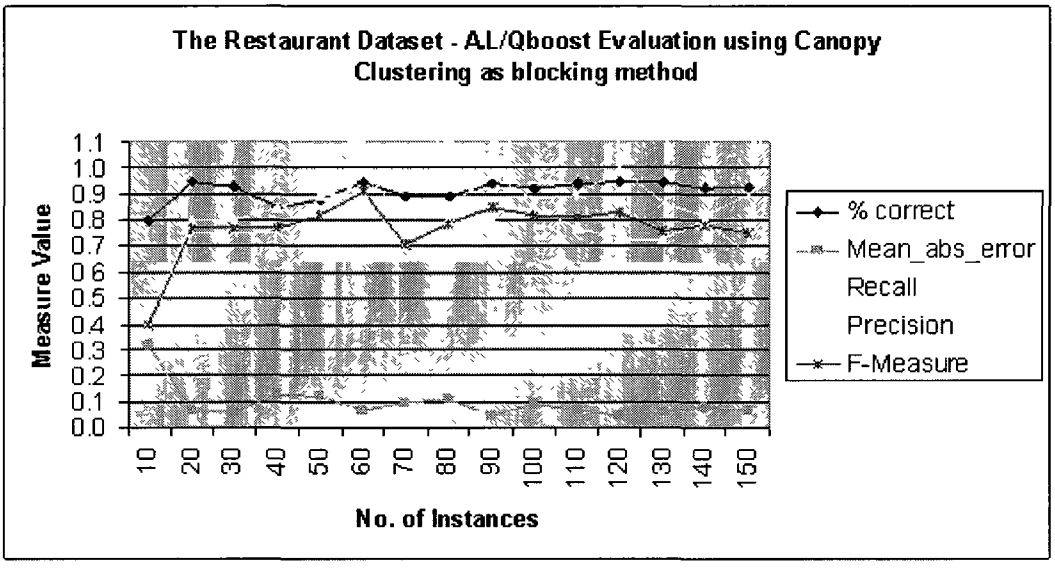


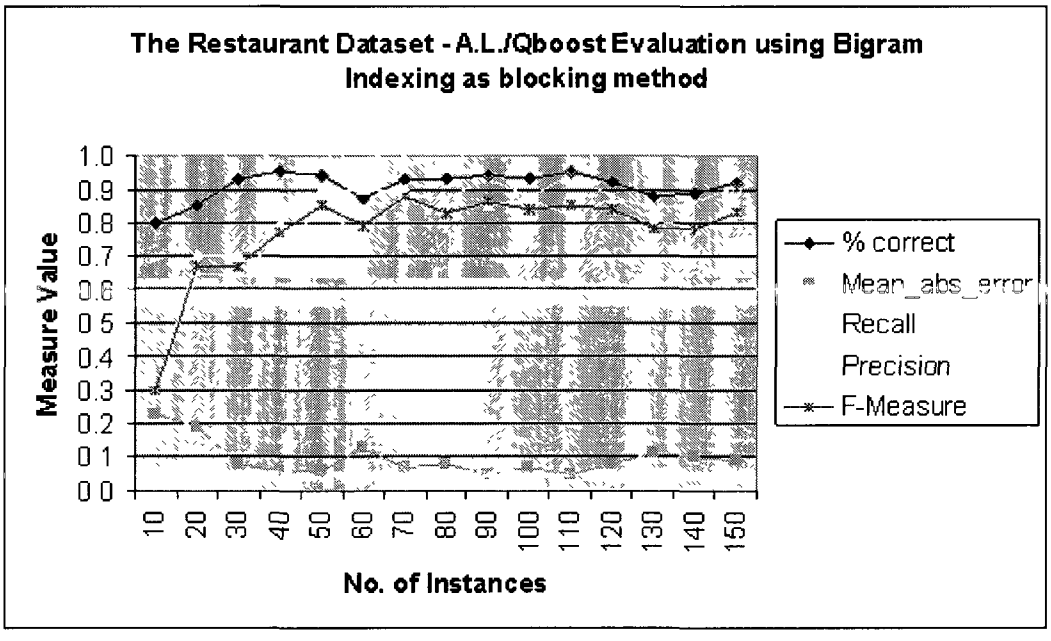
Figure 6.11: Evaluation using Canopy Clustering on the Mailing dataset when the dataset generates more errors($\approx$ 1000 records)

The evaluation on the three artificial datasets generated using the Mailing data generator, as described in Chapter 5, is depicted in Figures 6.8, 6.9 and 6.10. Recall that these datasets were generated by only varying the dataset size parameter and keeping as default all the other parameters about the amount and the types of error used to create duplicates. Graphs 6.8 and 6.9 show that the de-duplication function obtained using the original training sets of 10 instances gives the smallest performance values. However, as more instances are added, we see a progressive increase in performance. The increase is significant at the beginning, but stabilizes later. When there is no more significant increase as new instances are added, it means the training is done and there is no more challenging instances that the classifier has not seen yet. The evaluation when the Mailing dataset has 5000 instances is a bit different. The performance measures generated high values from the get-go. We can see however that, at the beginning new instances may cause big variances, causing the performance of the function to decrease and then increase again. At this stage, the classifier is not fully trained on all the challenging instances. As the training datasets has more instances, the variance diminishes and the learning process stabilizes.

One more dataset was created using the Mailing dataset generator. The dataset has 1016 instances and the parameter controlling the probability of inserted error was



(a) Using Canopy Clustering



(b) Using Bigram Indexing

Figure 6.12: Evaluation using Canopy Clustering when two different blocking methods are used on the same dataset

increased to a higher value compared to the three previous examples. Figure 6.11 shows that Active Learning still performs efficiently on an artificial dataset even in presence of more errors.

Finally, we compare how well Active Learning performs when the Canopy Clustering and the Bigram Indexing blocking methods are used on the same dataset. Figure 6.12(a) and Figure 6.12(b) reveal the results of both evaluations. The evaluation of the de-duplication function using both blocking methods produces very similar results. The classifier starts poorly, but after training on 10 instances, good results start to show and the results remain good throughout the training. When Bigram Indexing is used, the performance measures start with a value of only 0.3, which is 0.1 value less than when using Canopy Clustering. However, its performance rises quickly to reach close result values to Canopy Clustering. This is true after 40 more instances are appended to the training set. The Bigram Indexing appears to facilitate the learning process better than the Canopy Clustering does. As mentioned in Section 6.2.1, the reason may be that Canopy Clustering creates overlapping blocks of potential duplicates which may cause the unlabeled dataset to have multiple instances that have close distances.

6.2.3 Comparison of Query by Bagging and Query by Boosting

In this section, we compare the results of using Query by Bagging versus Query by Boosting with Active Learning for the task of de-duplication. We compare their behavior in different situations, using the same datasets used earlier. We particularly examine how the two methods perform when different blocking methods are used and when a real world dataset is used (see Figures 6.13 and 6.14). We also compare the two methods on datasets of different sizes and when an artificial dataset is used (996 records versus 4949 records).

Discussion

Figure 6.13 compares Active Learning with Query by Bagging and Active Learning with Query by Boosting when the Canopy Clustering method is used. By looking at the evaluation graph, we observe that the performance of Query by Bagging (Figure 6.13(a)) starts low. The performance of the learning function stays low until around 60 instances are added to the training set. Query by Bagging learns slowly compared to Query by Boosting (Figure 6.13(b)). As a matter of fact, Query by Boosting reaches high performance very quickly, after only 10 additional instances are added to the starting training set of 10 instances. Similarly, we compare the two methods when the Bigram Indexing is used as the blocking method (Figure 6.14). Once again, using Query by Boosting seems to yield better results than when Query by Bagging is used. Even

though Query by Bagging reaches good performance early on, the de-duplication function is not stable at the beginning. New instances cause the performance curve to oscillate. However, later during the learning process, the curve starts to stabilize. That is, the learning process is more steady with Query by Boosting.

Independently of the blocking method used on the Restaurant dataset, using Query by Boosting generates better performance faster than using Query by Bagging. This can be explained by the fact that Query by Boosting, which uses the Adaboost algorithm, has the advantage of being able to boost the performance of weak classifiers [65].

Next, we compare the two learning methods on datasets of different sizes. From Figure 6.15 where the dataset has 996 records, Active Learning with Query by Bagging shows a steady improvement in the performance of the de-duplication function as new instances are added to the training set. For Query by Boosting, the progress in performance is not as steady as Query by Bagging, but nevertheless, it reaches good performances after only 40 instances are appended to the training set. In Figure 6.16, the Mailing dataset with 4949 records is used to compare the two learning algorithms. Query by Bagging has low performance values at the beginning but then gradually improves as new instances are added to the training set. Query by Boosting, on the other hand, shows high performances from the start. Even though, at the beginning new instances may cause the performance to decrease and then increase again, the performance stays in the good performance range.

For both datasets (996 and 4949 records), the two methods seem to reach high performances early. By the time the training set has 50 instances, high values are already reached. Note also that in both datasets, Query by Boosting keeps increasing the performance and by the time the training set has around 100 instances, the performance values are close to 0.9.

As for the real world dataset (Figures 6.13 and 6.14) versus artificial dataset (Figures 6.15 and 6.16), there is not a big difference between using Query by Bagging or Query by Boosting with Active Learning. In both cases, adding new instances to the training set at the beginning varies from increasing and decreasing the performance. As more instances are labeled and put into the training set, the performance stabilizes.

6.3 Summary

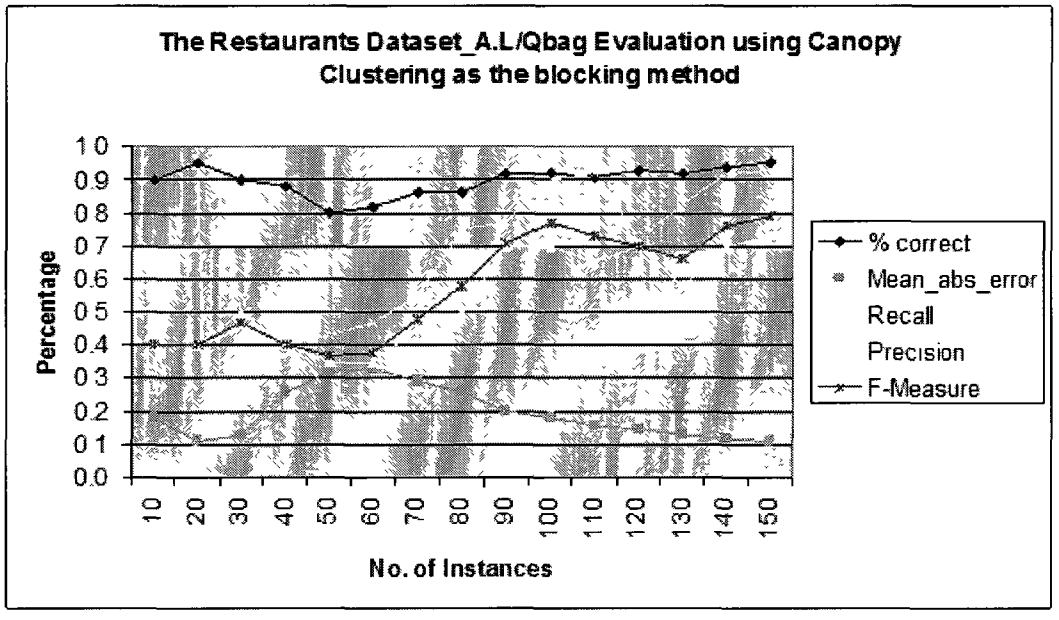
In this chapter, we presented the evaluation methods used, as well as the results of evaluating the de-duplication function when Active Learning is used with Query by Bagging and Query by Boosting. For both cases, we studied the effect of running the

function on synthetic datasets, real-world datasets, datasets of different sizes and using two different blocking methods.

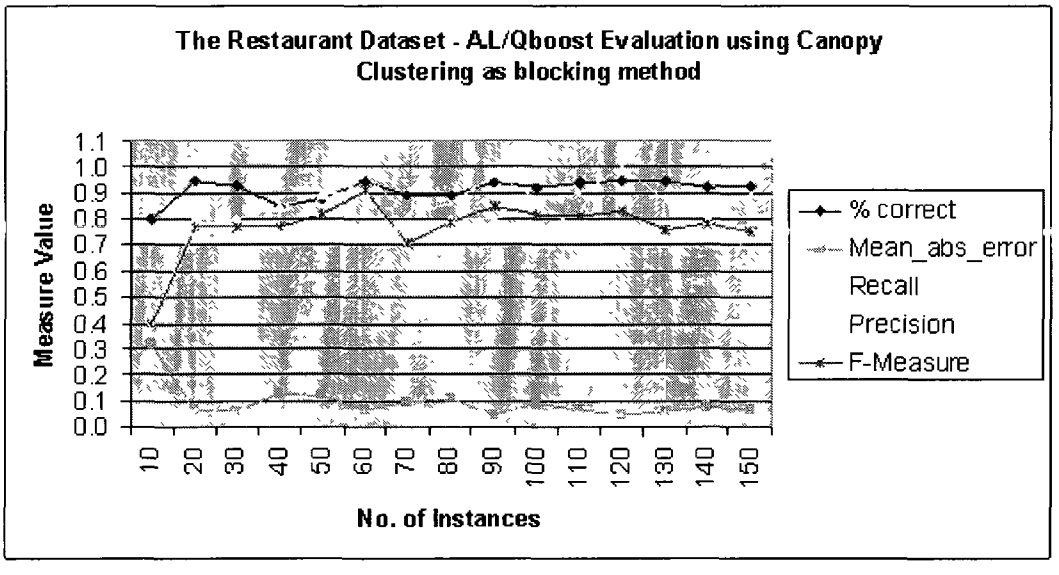
For Query by Bagging, the results showed that the de-duplication function produced good results when run on synthetic datasets. A steady improvement on the performance was observed, as Active Learning gradually and selectively constructed the training set used to teach the learner. On real-world datasets, it was observed that using Query by Bagging generally generates high performance except when noise is present in the dataset, in which case the performance of the function is negatively affected. For the size of the dataset, the experiment showed that small datasets had a steady and linear improvement in performance. Large datasets also yield high performance but the learning curve was less smooth than for smaller datasets. When Canopy Clustering and Bigram Indexing were compared using Query by Bagging, Bigram Indexing showed better learning results.

For Query by Boosting, both synthetic and real-world datasets generated good results and only required few informative instances to be added to the training set for high performance. However, for the real-world dataset, if the dataset contains noise, poor results in the learning process was observed. The size of the datasets did not seem to have a significant impact on the learning process, as high performances were observed each time. Using Query by Boosting also did not show any significant difference between using Canopy Clustering and Bigram Indexing.

Finally, we compared Query by Bagging and Query by Boosting. The comparison was conducted on two different blocking methods, on a small dataset and a large dataset and finally, on synthetic dataset versus real-world dataset. The experiment shows that independently of which blocking method used, Query by Boosting performed better than Query by Bagging. The comparison using different sizes showed similar results for both methods, with high performances observed. Finally, Query by Bagging and Query by Boosting had similar results on both synthetic datasets and real-world datasets where both produced high performances during the learning process.

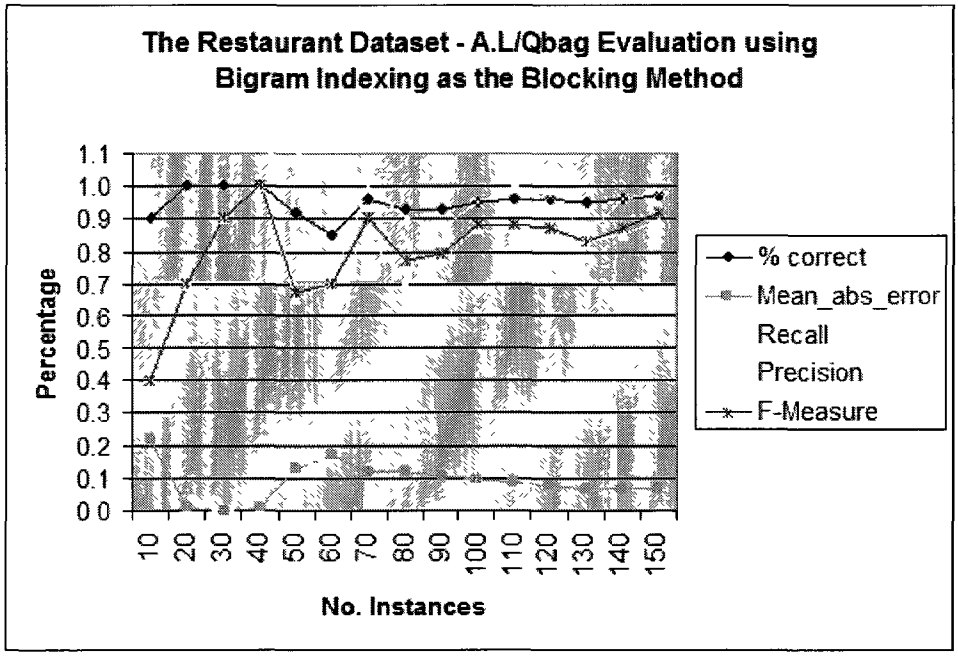


(a) Using Query by Bagging

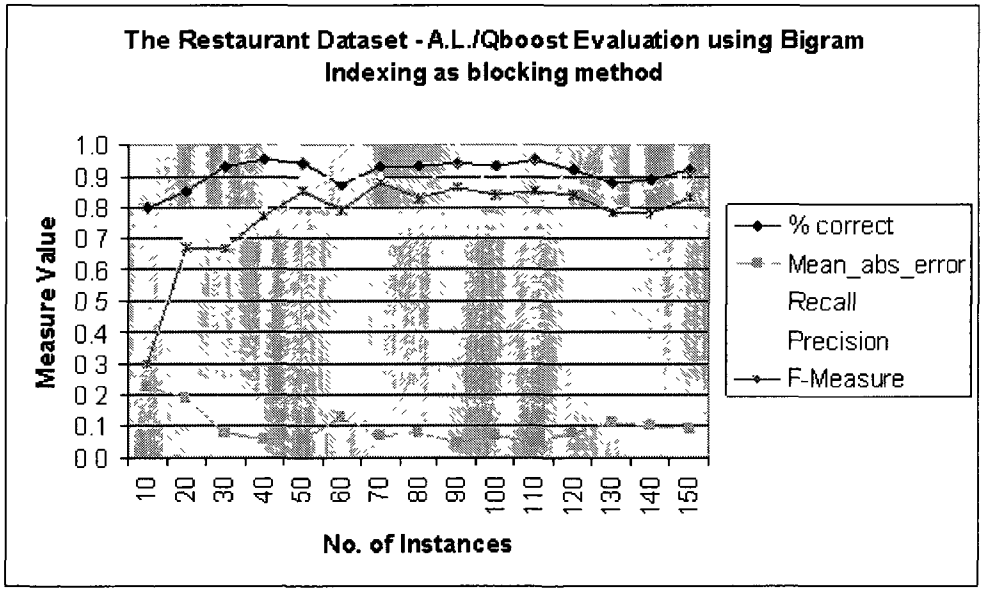


(b) Using Query by Boosting

Figure 6.13: Comparison of Qbag and QBoost with Canopy Clustering on the Restaurant dataset

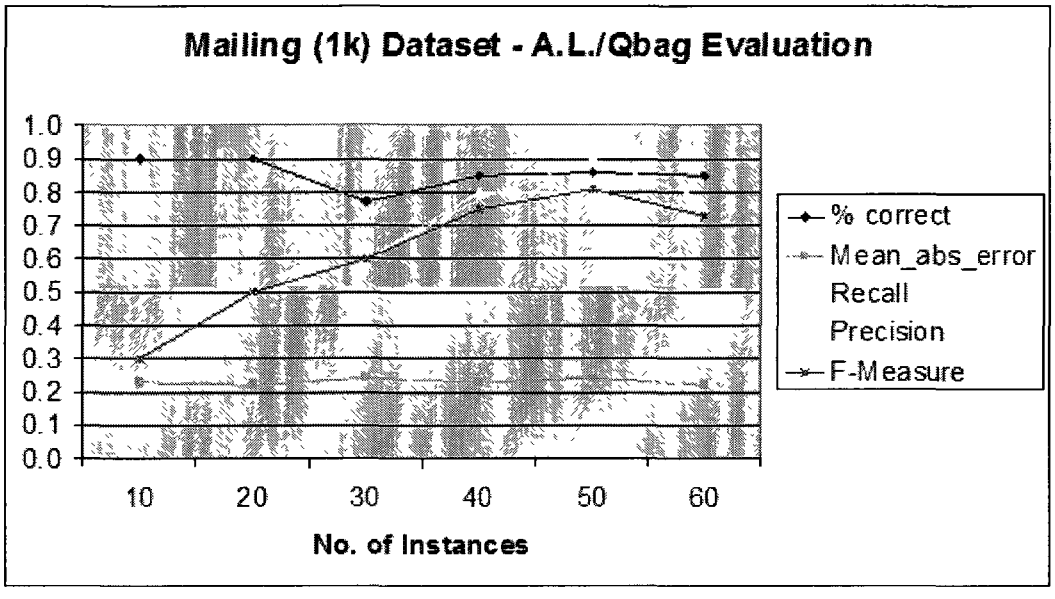


(a) Using Query by Bagging

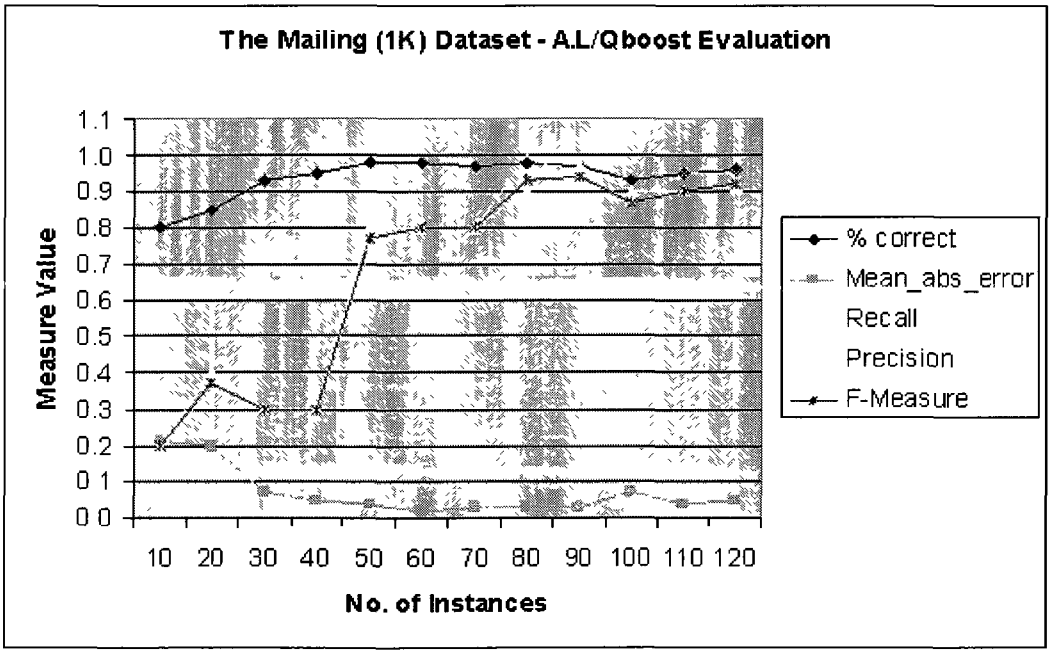


(b) Using Query by Boosting

Figure 6.14: Comparison of Qbag and QBoost with Bigram Indexing on the Restaurant dataset

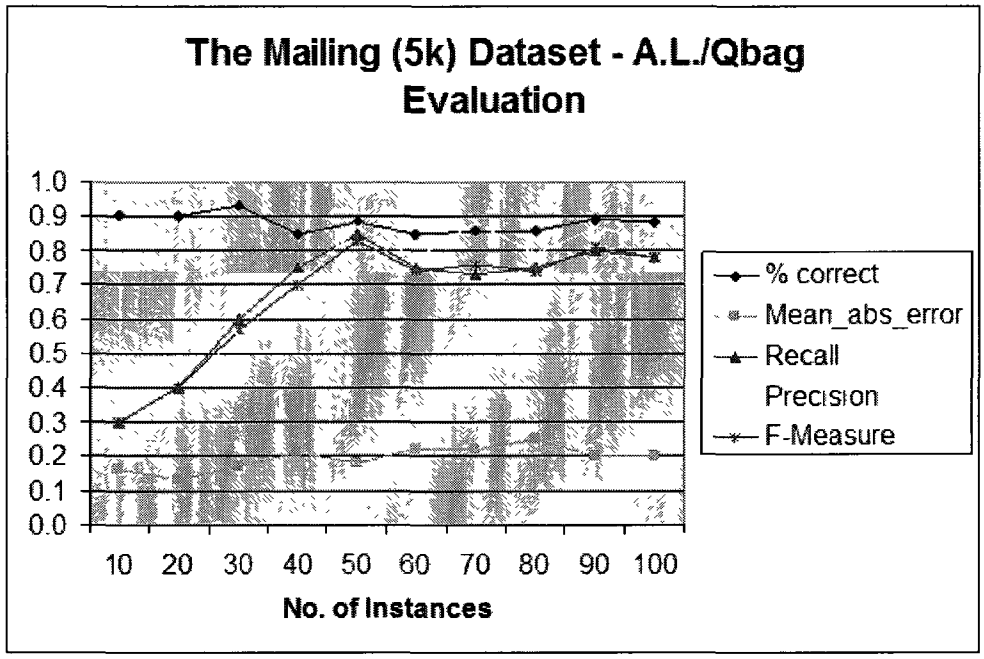


(a) Using Query by Bagging

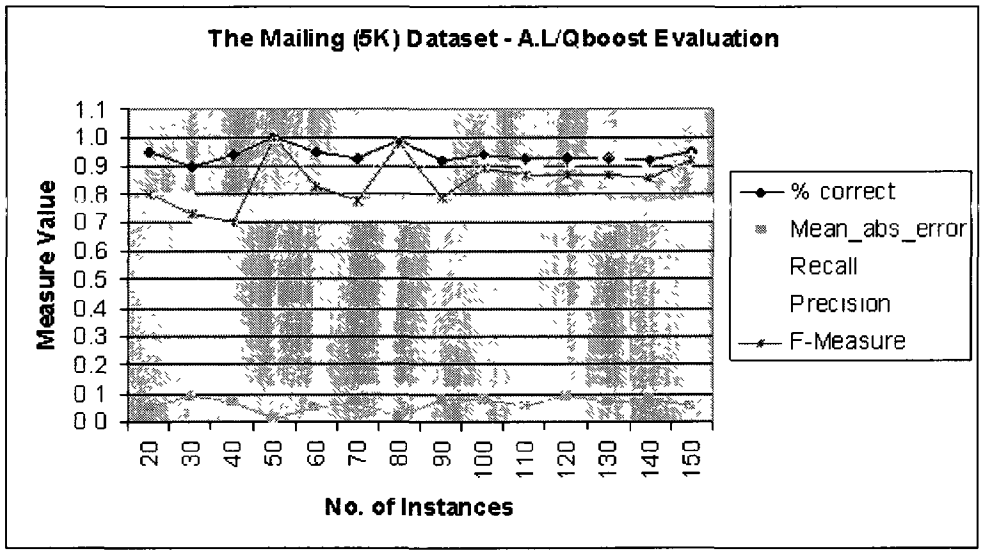


(b) Using Query by Boosting

Figure 6.15: Comparison of Qbag and QBoost with Canopy Clustering on the Mailing Dataset ($\approx$ 1000 records)



(a) Using Query by Bagging



(b) Using Query by Boosting

Figure 6.16: Comparison of Qbag and QBoost with Canopy Clustering on the Mailing Dataset ($\approx$ 5000 records)

Chapter 7

Conclusion

Active Learning is an iterative supervised learning which repeatedly chooses new informative instances and requests a user's input as to the class label of the chosen instances. These new instances are appended to the training set which is used to re-train a classifier. This technique reduced the number of training instances needed for classifiers to yield good predictive models. In this thesis, techniques of Active Learning were used in the task of detecting duplicates in a dataset.

This thesis provided a detailed explanation of different techniques used in the community of data mining that are of interest in our task of duplicate detection. We have shown that detecting duplicates using Active Learning requires that the data be prepared first before Active Learning is performed. The steps of preparing the data involve *attribute selection, blocking and usage of similarity functions.*

Attribute selection will ensure that only attributes that have an impact on the performance of the learned model are selected. Once the attribute selection is performed, blocking is performed. Since records in the dataset will need to be paired in order to find all duplicates, blocking methods avoid the need to pair-up all the instances, which would be very costly. Instead, the blocking methods use cheap distance functions and put only those instances that can be *potential* matches in the same block. Finally, similarity functions are run on pairs of instances in the same block. This will produce a new instance made of distance values. The identifier is then the record numbers of the two instances paired-up.

Once the data preparation phase is completed, the training set is created by randomly selecting few instances and the Active Learning phase begins.

7.1 Thesis Contributions

Duplicate Detection using Active Learning has been investigated before in the research communities. Our main contribution was to evaluate duplicate detection when two particular types of Active Learning are used, namely the Query by Bagging and the Query by Boosting methods. These methods use two ensemble learning methods, Bagging and Boosting respectively, which use re-sampling techniques to create diverse ensembles. Bagging works by re-sampling data with replacement, thus creating different subsets of the same size as the original dataset. To create ensembles, a learning algorithm is then run on each subset. On the other hand, we used the Adaboost algorithm as our boosting method. Adaboost creates ensembles by maintaining a certain weight for each instance. After the classifier is trained and a hypothesis is created, the weight of the misclassified instances is increased so that the focus is directed to those instances. The classifier is subsequently re-trained creating another hypotheses, and so on.

Once Query by Bagging and Query by Boosting have created ensembles, Active Learning selectively chooses instances that cause most disagreement among the ensembles. These instances are presented to a domain expert for labeling and appended to the training set. The classifier is retrained using the new training set.

The experiment we have conducted to evaluate the two types of Active Learning has showed that, in general, both techniques only require few good instances to be added to the training set to realize high performance. When comparing de-duplication using Active Learning on real-world data versus artificial data, the results showed that the de-duplication function was learned faster and yield better results on the artificial data. The results were also good for one of the real-world datasets, even though it took a bit more instances for the learning process to stabilize. The other real-world dataset yielded poor performance when both Query by Bagging and Query by Boosting were used, especially for the Query by Boosting. The reason is because the dataset had noise and Query by Boosting usually does not perform well in presence of noise. The results of the evaluation also showed that changing the size in the artificial dataset, as well as generating more errors in the artificial dataset, did not affect the good performance of the de-duplication function.

7.2 Future Work

Our work mainly focused on evaluating how well Query by Bagging and Query by Boosting performed when used in Active Learning for the task of finding duplicates in datasets. The emphasis was particularly put on the "Active Learning" part of the de-duplication

process. As far as the author is aware, this is the first study that evaluates the performance of Active Learning, based on Query by Bagging and Query by Boosting, for data de-duplication.

There exists more possible directions this work can take. One possible direction would be to compare Query by Bagging and Query by Boosting when different similarity functions are used. In fact, for our work, we only used one similarity function that is known to be good for strings of various sizes. It would be interesting to study how the two methods compare when different similarity functions are used, for the same attributes. and also for different attributes. For instance, we could examine the outcome of using multiple similarity functions on the same attribute to capture different types of distance information. For instance, if an attribute contains a name, we could use Soundex to capture the errors that are related to the phonetics and the Levenshtein edit distance to capture the errors related to the position of characters.

Another suggestion for future work would be to go beyond these two types of Active Learning. It would be worthwhile to study how new ensemble learning algorithms would perform when given the task of finding duplicates in different datasets. For example, ensemble learning algorithms such as Stacked Generalization or DECORATE can be investigated to see how Active Learning based on these methods would behave when given the task of finding duplicates in datasets [81, 56].

Bibliography

[1] Naoki Abe and Hiroshi Mamitsuka. Query Learning Strategies Using Boosting and Bagging. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 1 9, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[2] Charu C. Aggarwal. Towards Systematic Design of Distance Functions for Data Mining Applications. In *KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 9–18, New York, NY, USA, 2003. ACM.

[3] Eric Bauer and Ron Kohavi. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36(1-2):105–139, 1999.

[4] Rohan Baxter, Peter Christen, and Tim Churches. A Comparison of Fast Blocking Methods for Record Linkage. In *Proceedings of the Ninth ACM SIGKDD'03 Workshop on Data Cleaning, Record Linkage and Object Consolidation*, pages 25–27, Washington, DC, USA, August 2003.

[5] Mikhail Bilenko and Raymond J. Mooney. Learning to Combine Trained Distance Metrics for Duplicate Detection in Databases. Technical Report AI 02-296, Artificial Intelligence Laboratory, University of Texas at Austin, Austin, TX. feb 2002.

[6] Mikhail Bilenko and Raymond J. Mooney. Adaptive Duplicate Detection using Learnable String Similarity Measures. In *KDD '03: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 39–48, New York, NY. USA, 2003. ACM.

[7] Mikhail Y. Bilenko. *Learnable Similarity Functions and Their Applications to Record Linkage and Clustering.* PhD thesis, The University of Texas at Austin, 2006.

[8] Hendrik Blockeel. Maurice Bruynooghe, Sašo Džeroski, Jan Ramon, and Jan Struyf. Hierarchical Multi-Classification. In Sašo Džeroski, Luc De Raedt, and Stefan Wrobel. editors, *MRDM '02: Proceedings of the First SIGKDD Workshop on Multi-Relational Data Mining*, pages 21–35. University of Alberta. Edmonton, Canada. July 2002.

[9] Hendrik Blockeel, Leander Schietgat, Jan Struyf, Sašo Džeroski, and Amanda Clare. Decision Trees for Hierarchical Multilabel Classification: A Case Study in Functional Genomics. In *PKDD '06: Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 18–29, 2006.

[10] Leo Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140. 1996.

[11] Leo Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California at Berkeley, 1996.

[12] Leo Breiman. Heuristics of Instability and Stabilization in Model Selection. *The Annals of Statistics*, 24(6):2350–2383, 1996.

[13] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

[14] Abhirup Chatterjee and Arie Segev. Data manipulation in heterogeneous databases. *SIGMOD Record*, 20(4):64–68, 1991.

[15] Amanda Clare and Ross D. King. Knowledge Discovery in Multi-Label Phenotype Data. In *In: Lecture Notes in Computer Science*, pages 42–53. Springer, 2001.

[16] Amanda Clare and Ross D. King. Machine Learning of Functional Class from Phenotype Data. *Bioinformatics*. 18(1):160–166. 2002.

[17] William Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A Comparison of String Distance Metrics for Name-Matching Tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pages 73–78. Acapulco. Mexico, 2003.

[18] William Cohen and Jacob Richman. Learning to match and cluster entity names. In *ACM SIGIR'01 workshop on Mathematical /Formal Methods in IR, 2001.*, 2001.

[19] William Cohen and Jacob Richman. Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. In *KDD '02: Proceedings of the eighth*

ACM SIGKDD international conference on Knowledge discovery and data mining, pages 475–480, New York, NY, USA, 2002. ACM.

[20] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Mach. Learn.*, 15(2):201–221, 1994.

[21] Matt Culver, Deng Kun, and Stephen Scott. Active Learning to Maximize Area Under the ROC Curve. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 149–158, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Sanmay Das. Filters, Wrappers, and a Boosting-Based Hybrid for Feature Selection. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 74 81, Williamstown, MA, June 2001.

[23] Manoranjan Dash and Huan Liu. Feature Selection for Classification. *Intelligent Data Analysis*, 1(3):131- 156, 1997.

[24] Lee R. Dice. Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3):297 302, 1945.

[25] Thomas G. Dietterich. Machine Learning Research: Four Current Directions. *AI Magazine*, 18(4):97–136, 1997.

[26] Thomas G. Dietterich. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. *Machine Learning*. 40(2):139–157, 2000.

[27] Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *MCS '00: Proceedings of the First International Workshop on Multiple Classifier Systems*, pages 1–15, London, UK, 2000. Springer-Verlag.

[28] Thomas G. Dietterich. Ensemble Learning. In *The Handbook of the Brain Theory and Neural Networks, Second Edition*, pages 405–408, Cambridge, MA, 2002. The MIT Press.

[29] Mohamed Elfeky, Vassilios S. Verykios, and Ahmed K. Elmagarmid. TAILOR: A Record Linkage Toolbox. In *In ICDE*, pages 17–28. IEEE Computer Society, 2002.

[30] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilio S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[31] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[32] Ivan P. Fellegi and Alan B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[33] Yoav Freund and Robert E. Schapire. Experiments with a New Boosting Algorithm. In *International Conference on Machine Learning.* pages 148 - 156, 1996.

[34] Yoav Freund, Sebastian H. Seung, Eli Shamir, and Naftali Tishby. Selective Sampling Using the Query by Committee Algorithm. *Machine Learning*, 28(2-3):133 168, 1997.

[35] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. RainForest - a Framework for Fast Decision Tree Construction of Large Datasets. In *VLDB'98: Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 416–427. Morgan Kaufmann, 1998.

[36] Osamu Gotoh. An Improved Algorithm for Matching Biological Sequences. *Journal of Molecular Biology*, 162(3):705–708, December 1982.

[37] Lifang Gu, Rohan A. Baxter, Deanne Vickers, and Chris Rainsford. Record linkage: Current practice and future directions. Technical Report 03/83, CSIRO Mathematical and Information Sciences, Canberra, Australia, April 2003.

[38] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*, chapter 7. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[39] Mauricio A. Hernández and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In *SIGMOG'95, Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data.* pages 127–138, New York, NY, USA. 1995. ACM.

[40] Mauricio A. Hernández and Salvatore J. Stolfo. Real-world Data is Dirty: Data Cleansing and the Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2(1):9 -37, 1998.

[41] Paul Jaccard. The Distribution of the Flora in the Alpine Zone. *New Phytologist*, 11(2):37 50, 1912.

[42] Matthew A. Jaro. Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of the American Statistical Association*, 84(406):414–420. June 1989.

[43] Matthew A. Jaro. Probabilistic Linkage of Large Public Health Data File. *Statistics in Medecine*, 14(5-7):491–498, 1995.

[44] Mahesh Joshi, George Karypis, and Vipin Kumar. ScalParC: A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets. *Parallel Processing Symposium, International*, 0:573. 1998.

[45] Piotr Juszczak and Robert P.W. Duin. Uncertainty Sampling Method for One-Class Classifiers. In *Proceedings of the ICML-2003 Workshop: Learning with Imbalanced Data Sets II*, pages 81–88, 2003.

[46] Ron Kohavi and George H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1-2):273–324. 1997.

[47] Sotiris B. Kotsiantis. Supervised Machine Learning: A Review of Classification Techniques. *Informatica*, 31(3):249–268, 2007.

[48] Eugene F. Krause. *Taxicab Geometry*. Dover, 1986.

[49] Pat Langley. Selection of Relevant Features in Machine Learning. In *Proceedings of the AAAI Fall symposium on relevance*, pages 140–144. AAAI Press, 1994.

[50] Patrick Lehti and Peter Fankhauser. A precise blocking method for record linkage. In *Data Warehousing and Knowledge Discovery*, pages 210–220, 2005.

[51] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

[52] 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.

[53] Ray Liere and Prasad Tadepalli. Active Learning with Committees for Text Categorization. In *In AAAI-97*, pages 591-596, 1997.

[54] John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In *Proceedings of DARPA Broadcast News Workshop*, pages 249 252, 1999.

[55] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *KDD '00: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 169–178, New York, NY, USA, 2000. ACM.

[56] Prem Melville and Raymond J. Mooney. Diverse Ensembles for Active Learning. In *ICML '04: Proceedings of the Twenty-First International Conference on Machine Learning*, page 74, New York, NY, USA, 2004. ACM.

[57] Alvaro E. Monge and Charles Elkan. The Field Matching Problem: Algorithms and Applications. In *Knowledge Discovery and Data Mining*, pages 267 270, 1996.

[58] Michael Nashvili. Decision Trees and Data Mining, 2004.

[59] Saul B. Needleman and Christian D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteine. *J Mol Biol*, 43(3):443 453, 1970.

[60] Howard B. Newcombe and James M. Kennedy. Record Linkage: Making Maximum Use of the Discriminating Power of Identifying Information. *Commun. ACM*, 5(11):563-566, 1962.

[61] Howard B. Newcombe, James M. Kennedy, John S. Axford, and Allison P. James. Automatic Linkage of Vital Records. *Science*, 130:954–959, oct 1959.

[62] Foster Provost and Tom Fawcett. Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions. In *In Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*. pages 43–48. AAAI Press, 1997.

[63] Ross J. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[64] Ross J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[65] Ross J. Quinlan. Bagging, Boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. pages 725–730, 1996.

[66] Pradeep Ravikumar and William W. Cohen. A Hierarchical Graphical Model for Record Linkage. In *AUAI '04: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 454–461, Arlington, Virginia, United States, 2004. AUAI Press.

[67] Eric S. Ristad and Peter N. Yianilos. Learning String-Edit Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):522–532, 1998.

[68] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter Methods for Feature Selection: a Comparative Study. In *IDEAL'07: Proceedings of the 8th international conference on Intelligent data engineering and automated learning*, pages 178–187. Berlin, Heidelberg, 2007. Springer-Verlag.

[69] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive Deduplication using Active Learning. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, New York, NY, USA, 2002. ACM.

[70] Robert E. Schapire. The Boosting Approach to Machine Learning: An Overview. In *O=Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

[71] Greg Schohn and David Cohn. Less is more: Active learning with support vector machines. In *Proc. 17th International Conf. on Machine Learning*, pages 839 846. Morgan Kaufmann. San Francisco, CA, 2000.

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

[73] John C. Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *VLDB'96: Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India.* pages 544–555. Morgan Kaufmann, 1996.

[74] Temple F. Smith and Micheal S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[75] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning Object Identification Rules for Information Integration. *Information Systems*, 26(8):607–633, 2001.

[76] Kai M. Ting and Ian H. Witten. Stacked Generalization: when does it work? In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 866 - 871. Morgan Kaufmann, 1997.

[77] William E. Winkler. The State of Record Linkage and Current Research Problems. *Technical Report*, 1999.

[78] William E. Winkler. Overview of Record Linkage and Current Research Directions. *Technical Report*, 2006.

[79] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, CA, 2nd edition, 2005.

[80] Ian H. Witten, Eibe Frank, Len Trigg, Mark Hall, Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical Machine Learning Tools and Techniques with Java Implementations. In *Proceedings of ICONIP/ ANZIIS/ANNES99 Future Directions for Intelligent Systems and Information Sciences*, pages 192–196. Morgan Kaufmann, 1999.

[81] David H. Wolpert. Stacked Generalization. *Neural Networks*, 5(2):241–259, 1992.

[82] Nong Ye, editor. *The Handbook of Data Mining*. Lawrence Erlbaum Associates. Mahwah, NJ, 2003.