Palestine Polytechnic University
Deanship of Graduate Studies and Scientific Research
Master of Informatics

Machine Learning Based Disambiguation of Author's Names in ORCID Citations

Submitted by:
Jumah Y. Sleeman
Supervisor:
Dr. Hashem Tamimi

Thesis submitted in partial fulfillment of requirements of the degree
Master of Science in Informatics
June, 2018

The undersigned hereby certify that they have read, examined and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled: **Machine Learning Based Disambiguation of Author's Names in ORCID Citations**, submitted by **Jumah Y. Sleeman** in partial fulfillment of the requirements for the degree of Master in Informatics.

Graduate Advisory Committee:

Dr. Hashem Tamimi (Supervisor), Palestine Polytechnic University.

Signature:_____ Date:_____

Dr. Sami Snaineh (Internal committee member), Palestine Polytechnic University.

Signature:_____ Date:_____

Dr. Mohanad Jaabari (External committee member), Hebron University.

Signature:_____ Date:_____

Thesis Approved

Dr. Murad Abusubaih Dean of Graduate Studies and Scientific Research Palestine Polytechnic University

Signature:_____ Date:_____

Declaration

I declare that the Master Thesis entitled **"Machine Learning Based Disambiguation of Author's Names in ORCID Citations"** is my original work, and hereby certify that unless stated, all work contained within this thesis is my own independent research and has not been submitted for the award of any other degree at any institution, except where due acknowledgement is made in the text.

Jumah Y. Sleeman

Signature:_____ Date:_____

Statement of Permission to use

I, the undersigned, hereby agree that the library of Palestine Polytechnic University may make this M.A thesis entitled **Machine Learning Based Disambiguation of Author's Names in ORCID Citations** available to borrowers under the library rules.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made. Any coping or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Jumah Y. Sleeman

Signature:_____ Date:_____

Dedication

I dedicate this dissertation to my family and to all those who aspire to add new ideas and knowledge to serve humanity worldwide.

Acknowledgement

I would like to take this opportunity to acknowledge all those who helped me during this thesis work. I would like to thank my supervisor Dr. Hashem Tamimi for introducing me to the world of machine learning, his valuable suggestions, outstanding and insightful guidance during the course of this thesis work which facilitated my achievement of this dissertation.

I would also like to thank all the faculty and staff at the Computer Science Department at the University of Palestine Polytechnic for their assistance during my master's course-work.

I would also like to thank Alquds Open University for the opportunity that they gave to me to do my master.

I would like to thank my friend Ibrahim M.Qdemat for his endless support and encouraging me to do my best.

Finally, I would like to thank my family for their continuous support, encouragement and patience without which this thesis could not have been concluded.

الملخص

يُعتبر إزالة الغموض عن أسماء المؤلف نوعًا من الارتباطات القياسية التي يتم تطبيقها على المستندات العلمية. غالبًا ما يحدث الغموض بسبب عوامل مختلفة مثل المؤلف الذي له أكثر من اسم واحد أو مجموعة من المؤلفين الذين يشتركون في الاسم نفسه أو اختصارات الأسماء أو الأسماء المستعارة والأخطاء الإملائية (أخطاء في كتابة هذه الأسماء). لذلك من الصعب التمييز بين مؤلفي المستندات العلمية أو سيكون من الصعب تجميع الوثائق العلمية من قبل المؤلفين. إن تقنية التصنيف هي واحدة من أفضل الحلول للتعامل مع هذا التحدي حيث يتمثل الهدف من التصنيف في معالجة تحديد هوية المؤلف من خلال تحديد جميع المستندات التي تنتمي إلى مؤلف معين وتمييزها عن المصنفات أو المنشورات للمؤلفين الآخرين الذين يتشاركون في نفس الاسم. ومع ذلك ، فإنه لا يزال يمثل تحديا كبيرا نظرا لحجم المكتبات الرقمية المتزايد باستمرار. لذلك ، أصبحت نماذج التصنيف المتطورة حاسمة للغاية لتوفير حل تلقائي عالي الكفاءة. إن هذه الدراسة تهدف إلى توفير أفضل حل لمشكلة الغموض في أسماء المؤلفين مع الإشارة إلى استشهادات قواعد بيانات أوركد التي تتألف من أسماء المؤلفين المشاركين وعنوان مجلة البحث وعنوان البحث وسنة النشر من خلال إيجاد جميع الروابط المتعلقة بنفس المؤلف وتجميعهم معًا. والأهم من ذلك ، تم إجراء دراسة مقارنة بين مجموعة من مناهج التعلم الآلي ، مثل: خوارزمية البحث الثنائي ومنهجية الشبكات العصبية ومصنف بايز وخوارزمية الغابات العشوائية. لقد أثبتت نتائج التجربة أن تصنيف الغابات العشوائية دقيق بنسبة ٩٥ بالمئة تقريبًا في التحقق مما إذا كانت أزواج الاقتباس تنتمي إلى المؤلف نفسه أم لا. بالإضافة إلى ذلك ، كانت ميزة أسماء المؤلفين المشاركين الأكثر أهمية مقارنة بالحالات الأخرى التي لها تأثير بنسبة ١٢.٩ بالمئة في إزالة الغموض عن أسماء المؤلفين.

Abstract

Author's Names Disambiguation (AD) is a type of record linkage which is applied to scholarly documents. The ambiguity often occurs due to different factors such as authors who have more than one name version, or group of authors who share the same name. Therein, it is difficult to distinguish between scholarly document authors or to group scholarly documents by authors. Machine learning techniques provide a solution to deal with this challenge by training the machine to classify all the documents belonging to a certain author and distinguish them from works of other authors sharing the same name. However, AD is still a great challenge due to the ever-increasing size of digital libraries and the lack of training examples that represent the whole domain. This study aims at providing a solution by using ORCID citations as a large and reliable source of training data. A comparison study has been made among a group of machine learning approaches including j48, DNN, Naive Bayesian and Random forest. The results from the experiment have proven that Random forest classifier is the best among them with almost 95% accuracy. In addition, coauthors feature was the most important instance compared to the other instances which has an impact of 12.9% in eliminating ambiguity in author's names.

Contents

List of Tables

List of Figures

Glossary

A.C.C Author's Citation. 3

AC Author's Contribution. 3

AD Author's Names Disambiguation. vii, 1, 3, 25, 26, 31, 33, 44, 55

AM Author's Match. 4

AN Author's no-match. 4

AUC Area Under Curve. 47

C.M Confusion matrix. 14

CA Classification Accuracy. 15

CE Classification Error. 15

CM Citation Matching. 3

DL digital library. 2

DM Distance Matrices. 51, 52

DNN Deep Neural Networks. 47

F$_{num}$ The number of features to consider when splitting each node. 13

FPR False Positive Rate. 50

J48 Decision Tree. 47

LNFI Last-name First-intial. 4

MC Mixed Citation. 4

Orcid ID Author's Identifiers. 3

RF Random Forest. 12, 13, 47

ROC Receiver Operating Characteristic. 17

SC Split Citation. 4

T$_{depth}$ The tree depth. 13

T$_{num}$ total number of trees. 13

TPR True Positive Rate. 50

Weka Waikato Environment for Knowledge Analysis. 10

Chapter 1

Introduction

The quantity of scholarly documents is growing rapidly. Different authors may share the same name and may also publish under various names. This may lead to inconsistency in publications data where publications of some authors be placed under different authors. For the researchers in their fields, it is important to find all working activities of the same author and cluster them together. Our objective in this thesis is to introduce a solution to disambiguate the authors' names.

1.1 Problem Statement and Motivation

Author's Names Disambiguation (AD) is a very important and complex research topic. During search and retrieval of information there is a shortage in results because of the great potential of authors who share the same name which negatively influence the scientific research life cycle. Therefore, it is necessary to find a reasonable and effective way to distinguish the different authors who share the same name and thereby, identify the identity of each author.

Disambiguation of author's names can enhance the scientific discovery process and improve the efficiency of research funding and collaboration within the research community. It is important to clarify the ambiguity in the authors' names as a mean of gathering their research work, make their research known, increase the chance of citation, ensure that the research is counted in research assessment and to increase the chance of new collaboration. AD helps authors to have complete profiles containing all their work which increases the accuracy and reliability to distinguish and cluster their works.

AD is a challenging problem that is caused by multiple reasons. Among which, authors may publish under multiple names for variety of reasons including different spelling, misspelling, name change due to marriage, or the use of the family names and initials. Figure1.1 illustrates a case of AD problem. In the upper half of the figure, four different authors' share the same name "Mohammad Imran" while, in the lower half, the same author has four different names: The first name then the family name, the first initial of the first name then the family name, the family name followed by the first initial of the given name and finally, the family name followed by the

first name.

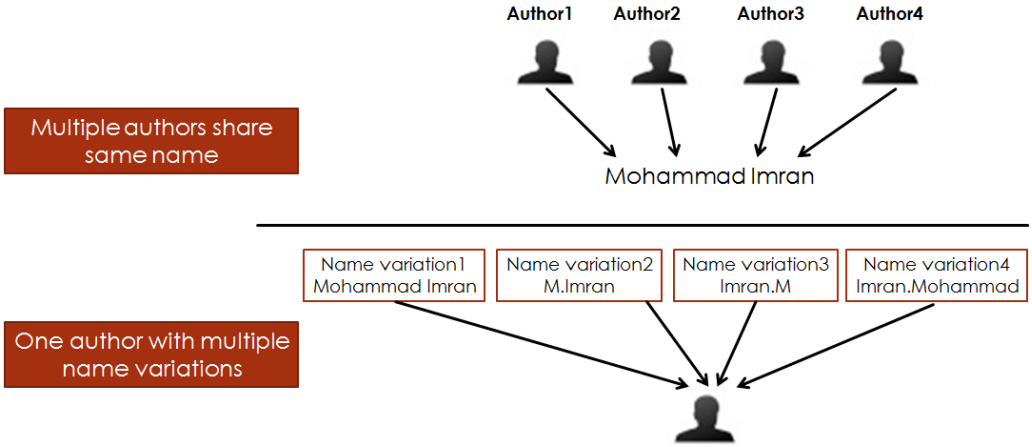

Figure 1.1: Name Disambiguation

Typical approaches for AD rely on information about the authors such as their affiliations, email addresses, year of publication, co-authors and other topic information to distinguish authors. Data that comes from individuals and organizations seems to be a hybrid system where all who participate in research, scholarship, and collaborations are uniquely identified and connected to their contributions and affiliations. In short, with a hybrid system designed to accommodate records from multiple sources, how do we determine that citation X record from Y different sources are likely to be referring to the desired researcher. These information can be used in machine learning approaches to decide whether two papers refer to the same author or not.

In recent years, the collection of bibliographic information relevant to a particular article has become a major challenge [2]. Users often need to exploit the relationships between citations to determine the impact of a particular article digital library (DL) [3]. In order to keep the citations of stored documents in DLs, consistent and up-to-date, we must keep in mind the excellent description of the four main challenges that impact name disambiguation that have been offered by (Smalheiser & Torvik, 2009) [4]

1. The name abbreviations that consist of the initial letter or parts of the first name, middle name and family name, are included in the articles database instead of the full-name.

2. Identical names in which multiple authors may share the same name label.

3. The same individual might write more than one name due to:

   - Spelling errors.
   - Name changes (for marriage, religious conversion, etc.).
   - Spelling variants.
   - The use of pen names.

4. Pseudonyms or alias that someone such as an author uses instead of his or her real name.

The Open Researcher and Contributor ID (ORCID) registry presents a unique opportunity to solve the problem of author names ambiguity [9]. From ORCID organization perspective, creating identifiers for individual researchers means creating a system where there is a one-to-one relationship between the identifier and the researcher. But every researcher could apply for ORCID identifier several times, this may mean many identifiers for the same author, uncompleted profile and that the author might write his name in different forms. Therefore, ORCID is not enough to solve the ambiguity of authors' names. ORCID mission statement implies that its system has the ability to generate a unique identifier for each author registered in the system. The questions about how each author profile records are created, how they are managed and corrected and who owns them are of a very importance and need to be resolved well before any public ORCID system is built and launched.

In this work we propose a new method to solve the AD problem using supervised machine learning. Like some existing literature, we re-define the AD problem as a binary classification problem. The classification system in our case reads the meta data of two authors and maps them to the $similar$ class or to the $notsimilar$ class. Also, we construct a web-application to predict wither two citations belong to same class or not as illustrated in appendix A.

Researchers who follow this approach has the challenge of collecting and labeling data for training the classifier. Our approach in this thesis is to make use of ORCID data for training since it represents a pre-labeled and representative dataset. This enabling us to build an accurate and reliable machine learning model that can produce a solution for the ambiguity problem in the names of authors without the effort needed to collect and label new data. ORCID contains citation data, each citation consists of the basic attributes namely, "coauthors names, journal title, work title and the publishing years".

1.2 Definitions

Below are the definitions and the acronyms that have been used through the thesis:

1. Author's Identifiers (Orcid ID): ORCID system provides a unique 16 digit identifier to authors'.

2. Orcid-profile : Each author has one profile with only one ORCID ID.

3. Author's Contribution (AC): The author's work information consist of Author's Affiliation (AF) and Author's Citation (A.C.C). AF or the name of the institution where the author work, consists of the organization name and the department name; while A.C.C contains the citation information.

4. Citation Matching (CM): Given two sets of publications X and Y, in order to find for each $x \epsilon X$ a set of $y_1, y_2, y_3, ..y_n, \epsilon Y$ such that both x and y_i $(1 \leq i \leq n)$ belong to the same

author. Because of the lack of fixed format for citations, various names may be attributed to a single author.

5. Mixed Citation (MC): Given a collection of publications X, by an author A_1, our goal is to identify publications by another author A_2 in X, when A_1 and A_2 share the same name string [5].

6. Split Citation (SC): Given two sets X and Y of author's names and associated publications, the challenge is to find each author's name $x \epsilon X$ in a set of author's names, $y_1, y_2, y_3, ..y_n, \epsilon Y$ such that both x and y_i $(1 \leq i \leq n)$ are names variants of the same author. [6].

7. Last-name First-intial (LNFI): The author's string name established from the first initial of the given name and the family name (e.g. Wang, Y).

8. LNFI Cluster: In which a group of authors share the same last-name first-initial, e.g., there are 829 Zhang,Y authors share LNFI.

9. Author's Match (AM): Two or more ACs that refer to the same individual.

10. Author's no-match (AN): Two ACs that refer to different individuals.

1.3 Thesis Structure

The remaining parts of this thesis are organized as follows: Chapter 2 presents the background for this thesis with a detailed description of the ORCID data structure, machine learning, performance evaluation techniques and the distance matrices. Chapter 3 discusses research related of this work. Chapter 4 contains the methodology and data analyses. It presents the process of selecting the ORCID data-sets, feature representations used by machine learning algorithms, and the general methodology used by the machine learning methods in predicting new citations records. Chapter 5 shows the experiments done to predict new citation pairs, random forest versus other machine learning approaches, and the results obtained from these experiments. Chapter 6 contains conclusion and future work. It summarizes the work done in this thesis.

Chapter 2

Background

2.1 ORCID Structure

ORCID is a non-profit organization, whose aim is to solve the author's name ambiguity problem in scholarly communications by creating unique identifiers (ORCID ID) for individual researchers and supports linkages between each author and his professional activities as illustrated in Figure 2.1. By using ORCID ID, ORCID ensures that all researchers activities and outputs are easily discoverable [7]. ORCID membership is opened to any organization interested in

Figure 2.1: Author profile [1]

integrating ORCID identifiers [8]. ORCID database records are created by individuals to whom the records refers, in which we can share information for both public and registered members to support open access to information for the research community. Figure 2.2 illustrates ORCID records structure which consists of:

1. Root element: The content of the researcher data is added within the root element.
2. ORCID identifiers (ORCID ID): ORCID system provides a unique sixteen digit identifier to each person, e.g. 0000-0002-4210-6896.
3. Language preference: English is the default language, but it can be changed into the user's preference.

4. History details: This part shows how the ORCID record was created. It can be either "website" or "direct". When the record was created through the ORCID registry directly, "member-referred", when a member sent a user to the Registry and the sign up was done as part of an author's connection and "API" when it was created with a batch create process on behalf of the user.

5. Biographical information: Both personal and biographical information are included under ORCID-bio, available under ORCID-profile. It contains: Personal-details, biography, external website links, contact-details, keywords and external identifiers.

6. ORCID Activities: It comes after the biographical information. It includes the affiliations, works and/or funding items. The affiliation section of an ORCID record is recorded under ORCID-activities. This area is separated from the ORCID-bio fields. Affiliation includes many fields as described in Figure 2.2. Works information which includes the author's activity work as illustrated in Figure 2.3

7. Source: Contains information about how and when the elements were added to the record. The elements are: ORCID-history, external-identifier, affiliation, ORCID-work and funding.

8. Create-Date: The date and time when the element was created.

9. Last modified date: The date and time when the record was last modified.

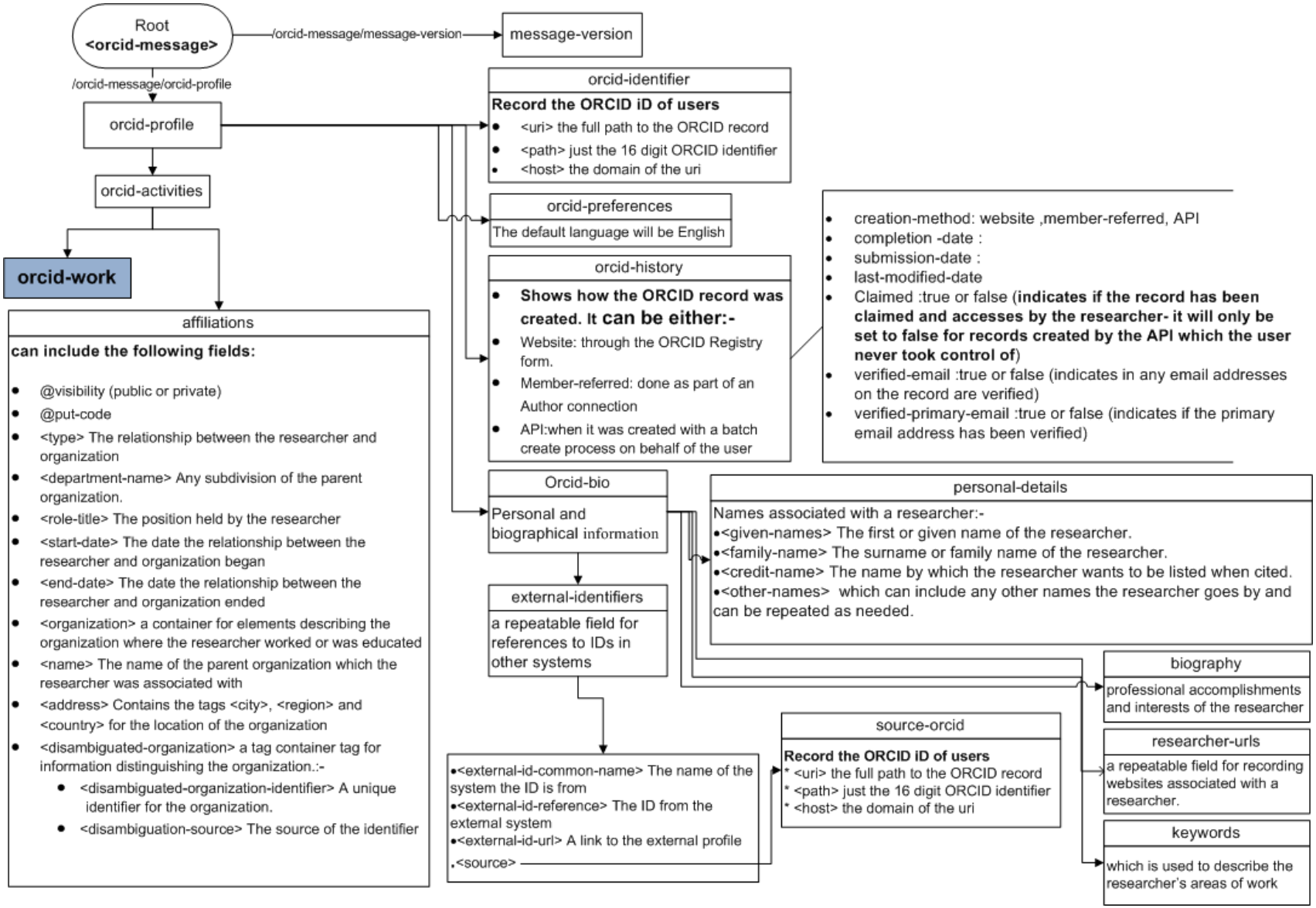

Figure 2.2: ORCID record structure

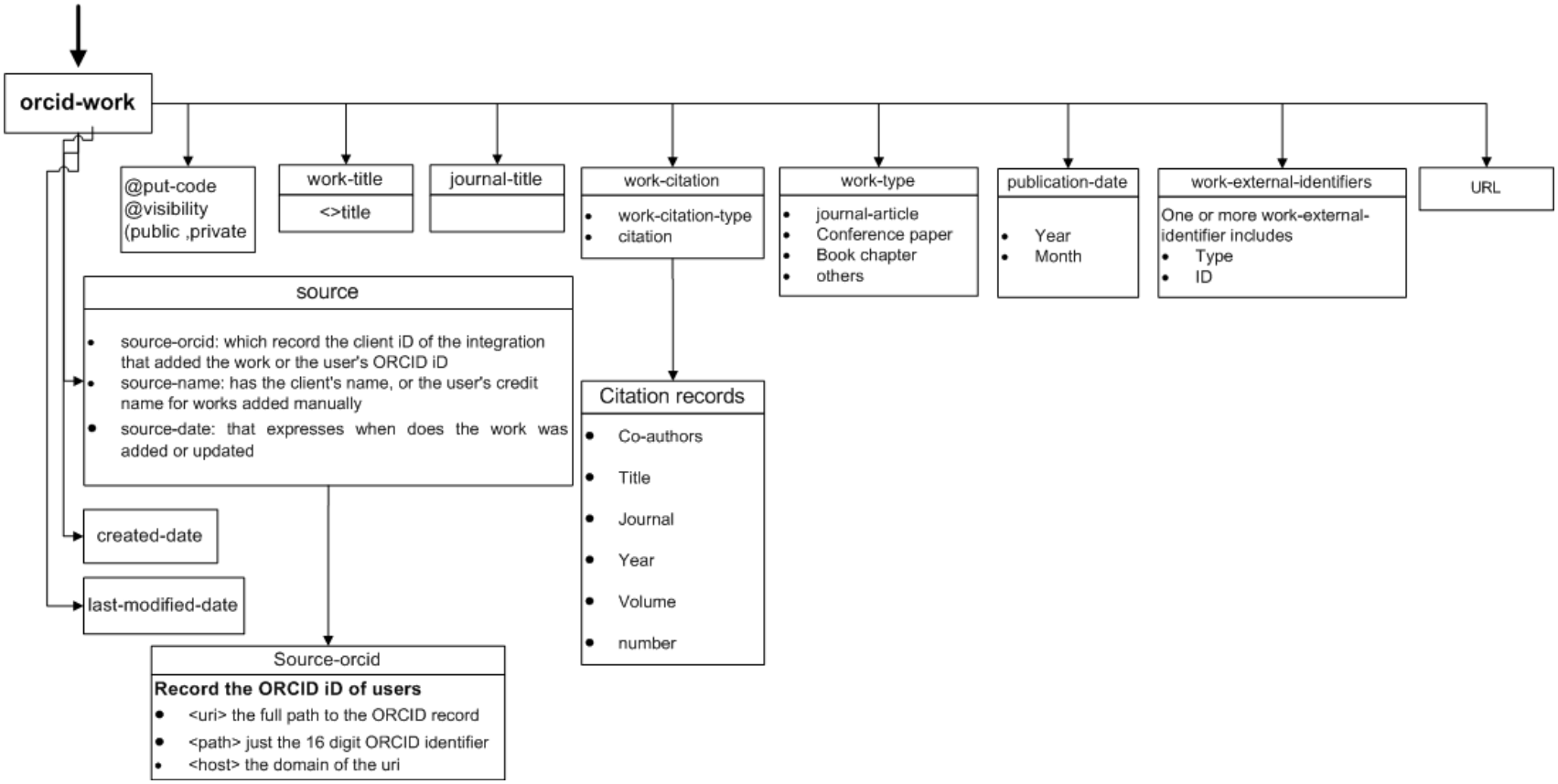

Figure 2.3: ORCID-work activities

In order to effectively identify and achieve this mission, ORCID needs to be a part of several publisher work-flows:

Firstly, *manuscript submission system* where ORCID identifier can be collected when an author submitting the basic information such as publication title, author's name, affiliation, abstract, cover letter, etc, then journals could ask coauthors to enter their ORCID identifiers.

Secondly, *searching for prior work* by linking author's name with ORCID identifier as illustrated in Figure 2.4. This will be used as a public ORCID ID to eliminate the ambiguity of searching taking into account the strategies of ORCID registration process as follows:

1. Non-active researchers could apply for ORCID identifiers several times.
2. The system would depend on a third-parties to provide organizationally-asserted records.
3. Control over profile information will determine that citation X from Y different sources refers to the desired researcher.

Figure 2.4: Public ORCID ID

Name ambiguity can exists clearly when the author's name is introduced by first-name initial and last-name. For example, different names Sung Jin Kim and Seon-Kyu Kim are simplified to the same name label S Kim [9]. In ORCID database, the string name "Wang Y" has various name variations, e.g, "Wang Ying", "Wang Ying Yang" and "Wang Lee Yung". As well in the web DBLP [10], the author page of "Yu Chen" contains citations from three different people with the same name: "Yu Chen" from University of California Los Angeles, "Yu Chen" from Microsoft at Beijing branch, and "Yu Chen" as the senior professor from Renmin University of China. This problem is identified as a situation where multi authors' share the same name label.

According to Smalheiser and Torvik, in ORCID files authors can be identified by the first initials only, by the middle name included with the family name, or by the family name only instead of the full name. Because of the growing orientation of different science approaches, it has become more difficult to tell whether John Smith publishing about linguistics is different from John Smith publishing in biochemistry, whereas in the past, two identities could be safely assumed. [4].

Therefore, it is not practical to manually create a profile for each author due to the presence of huge digital libraries. In other words, automatic name disambiguation project should consider all different cases to help others using data in term of performance, accuracy and availability.

2.2 Introduction to Weka

Weka is a collection of machine learning algorithms and data pre-processing tools. It is designed in a flexible ways so that you can try out existing methods on new and unseen datasets. It provides large-scale support for the whole process of experiments, including preparing the input data, evaluating learning schemes statistically, and visualizing the input data and the result of learning. As well as a wide variety of learning algorithms, it includes a wide range of pre-processing tools. This diverse and comprehensive toolkit is accessed through a common interface so that its users can compare different methods and identify those that are the most appropriate for the problem at hand.

Waikato Environment for Knowledge Analysis (Weka), is a Java based open-source data mining tool developed by the University of Waikato. In recent years, Weka has also been implemented in Big Data technologies such as Hadoop [11]. The system is written in Java and distributed under the terms of the GNU General Public License. It runs on almost any platform and has been tested under Linux, Windows, and Macintosh operating systems. It provides a uniform interface to many different learning algorithms, along with methods for pre-processing, post-processing and for evaluating the result of learning schemes on any given dataset.

Weka provides implementations of learning algorithms that you can easily apply to your dataset. You can pre-process a dataset, feed it into a learning scheme, and analyze the resulting classifier and its performance without writing any program code at all. The workbench includes methods for all the standard data mining problems: regression, classification, clustering, association rule mining, and attribute selection. Getting to know the data is an integral part of the work, and many data visualization facilities and data pre-processing tools are provided. All algorithms take their input in the form of a single relational table in the ARFF format, which can be read from a file or generated by a database query.

One way of using Weka is to apply a learning method to a dataset and analyze its output to learn more about the data. Another one is to use learned models to generate predictions on new instances. A third one is to apply several different learners and compare their performance in order to choose one for prediction. The learning methods are called classifiers, and in the interactive Weka interface you select the one you want from a menu. Many classifiers have tunable parameters, which you access through a property sheet or object editor. A common evaluation module is used to measure the performance of all classifiers. Implementations of actual learning schemes are the most valuable resource that Weka provides. But tools for pre-processing the data, called filters, come a close second. Like classifiers, you select filters from a menu and tailor them to your requirements. Weka also includes implementations of algorithms for learning association rules, clustering data for which no class value is specified, and selecting relevant attributes in the data [12].

2.2.1 Using Weka

When you start up Weka you have to choose among four different user interfaces: the *Explorer*, the *Knowledge Flow*, the *Experimenter*, and the *command-line* interface as shown in Figure 2.5.

Figure 2.5: Weka GUI Chooser

The easiest way to use Weka is through a graphical user interface called the *Explorer* as illustrated in Figure 2.6. This gives access to all of its facilities using menu selection and form filling. The *Explorer* interface presents choices as menus, buttons, text-boxes, presenting options as forms to be filled out. Helpful tool tips pop up as the mouse passes over items on the screen to explain what they do.

There are two other graphical user interfaces supported by Weka. The *Knowledge Flow* interface allows you to design configurations for streamed data processing. The *Knowledge Flow* interface lets you drag boxes representing learning algorithms and data sources around the screen and join them together into the configuration you want. It enables you to specify a data stream by connecting components representing data sources, pre-processing tools, learning algorithms, evaluation methods, and visualization modules. Weka's third interface, the *Experimenter*, is designed to help you answer a basic practical question when applying classification and regression techniques. Behind these interactive interfaces lies the basic functionality of Weka. This can be accessed in raw form by entering textual commands, which gives access to all features of the system [12].

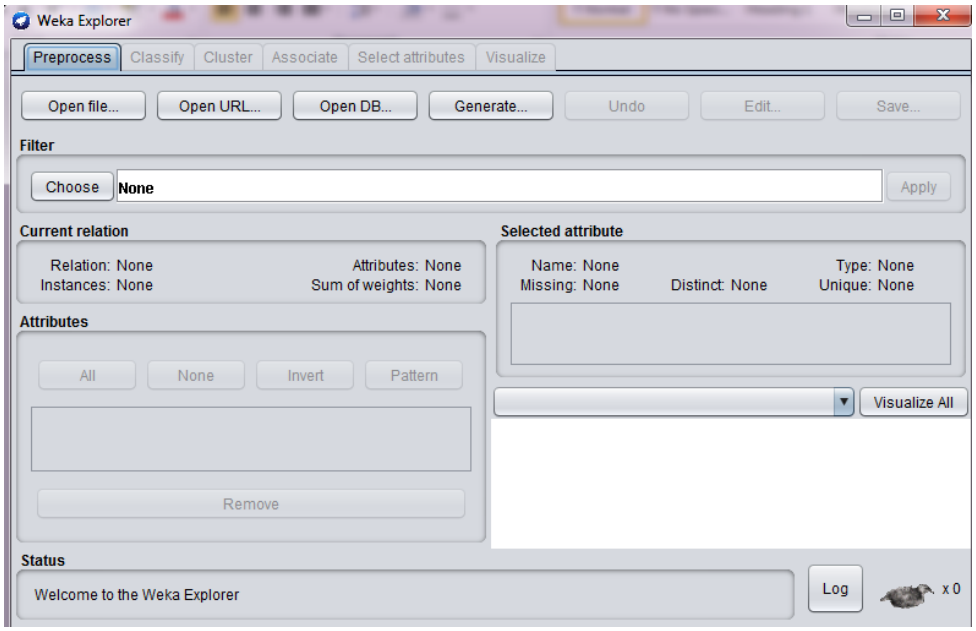

Figure 2.6: Weka Explorer

2.3 Random Forest Classification

Random Forest (RF) classification is one type of machine learning approaches. It belongs to the ensemble learning algorithm which received increasing interest because they are more accurate and robust to noise than single classifiers [13]. Ensemble algorithms are those which combine more than one algorithm of the same or different kinds for classifying objects.

RF was proposed by Breiman and presents many advantages. RF algorithm is a supervised machine learning algorithm which is capable of performing both regression and classification tasks. It also handles the missing values and maintains accuracy for missing data. Besides, it has the power to handle large data set with higher dimensionality. Moreover, it does not allow overfit the model and runs efficiently on large database. It estimates the important variables in the classification and generates an internal unbiased estimate of the generalization error (Out-of-bag error). Finally, it computes proximities between pairs of cases used in locating outliers and is relatively robust to outliers and noise. On the other hand, Breiman underlined some disadvantages of the RF. First of all, it does good job at regression but not as good as for classification. Secondly, the control the user has is very minimized on the model and it is like a black box approach for statistical models [13].

In RF, we create multiple decision trees $h(X, \Theta_k)$, where $h(\Theta_k)$ are the random vectors which are distributed identically and X is the variable to be classified. For constructing the k trees, RF first generates k random vectors $h(\Theta_1, \Theta_2, .., \Theta_k)$ which are independent of each other and of the same distribution. Each tree provides a classification based on the given training samples and random vectors Θ_i where $i \epsilon [1, k]$. RF saves the tree votes for specific class and it chooses the classification based on the majority of votes over all the other trees in the forest.

Equation 2.1 calculates the voting model of RF [14].

$$H_R(X) = max \sum_{i=1}^{k} I(h(X, \Theta_k)) \tag{2.1}$$

Where, $H_R(X)$ denotes the combination classification model and $I(h(X, \Theta_k)$ is the classification result of the decision tree on the input variables X. But in regression, it calculates the average of the outputs by different trees.
Supposed training set is given as : $[I1, I2, I3, I4]$ with corresponding labels as $[L1, L2, L3, L4]$, RF may create three decision trees taking input of subset for example

1. $[I1, I2, I3]$
2. $[I1, I2, I4]$
3. $[I2, I3, I4]$

Finally, RF predictions are based on the majority of votes from each of the decision trees. A single decision tree may be prone to a noise, but many decision trees reduce the effect of noise which gives more accurate results. Alternatively, the random forest can apply weight concept for considering the impact of result from any decision tree. Tree with high error rate is given low weight value and vice-versa. This would increase the decision impact of trees with low error rate.

2.3.1 RF Parameters

RF has number of parameters: The total number of trees (T_{num}) to be generated. The number of features to consider when splitting each node (F_{num}). The tree depth (T_{depth}) [15, 16]. Developing the correlation increases the forest error rate. Increasing the strength of the individual trees decreases the forest error rate inasmuch as a tree with a low error rate is a strong classifier.

In order to provide optimal balance between the correlation and the strength of the individual trees we must reduce the number of random attributes. For each decision tree on the random forests, the pruning is not necessary [14]. High accuracy and lower error rate are achieved from strong individual tree. According to Breiman [17] the default value for m is $m = \lfloor log_2(M) + 1 \rfloor$ or $m = \sqrt{M}$, where M is the total number of features and the default value of the tree depth is $T_{depth} = 0$.

2.4 Machine Learning Performance Evaluation

The metrics that we choose to evaluate the machine learning approaches are very important. Choice of metrics influences how the performance of machine learning approaches is measured and compared. Next sections will illustrate how to select and use different machine learning performance metrics.

2.4.1 Information Gain (InfoGain)

We evaluated the worth of an attribute by measuring the information gain with respect to the class. InfoGain is used to measure the dependence between features and labels and calculates the information gain between the i_{th} feature f_i and the class labels C [18]. Equation 2.2 is used to calculate the InfoGain.

$$InfoGain(f_i, C) = H(f_i) - H(f_i|C) \tag{2.2}$$

where $H(f_i)$ is the entropy of f_i and $H(f_i|C)$ is the entropy of f_i after observing C as illustrate in Equations 2.3 and 2.4

$$H(f_i) = -\sum_j p(x_j) log_2(p(x_j)) \tag{2.3}$$

$$H(f_i|C) = -\sum_k p(c_k) \sum_j p(x_j|c_k) log_2(p(x_j|c_k)) \tag{2.4}$$

In information gain, a feature is relevant if it has a high information gain. Features are selected in a univariate way, therefore, information gain cannot handle redundant features [19].

2.4.2 Cross Validation for Classification Problems

Cross validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it [20]. In k-fold cross-validation, the original data is randomly partitioned into k equal size subsamples. From k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once [21, 22].

2.4.3 Model Evaluation Metrics

An evaluation metric is always needed to go along with cross validation which depends on the type of the problem we are addressing [23]. Confusion matrix (C.M): Is defined as a table that describes the performance of a classification model. We can think of C.M as a table of the two types of correct predictions that the classifier can make as well as a table of the two types of incorrect predictions. Every observation in the testing set is represented exactly in one box of the C.M. Sometimes, the C.M will be explicitly labeled with the total number of represented observations.

The size of C.M depends on the classifier problem, e.g, two by two C.M size represents the binary classification, therefore, if there were five possible response classes, this would be a five

Table 2.1: Confusion matrix (C.M)

		Actual result/Classification	
		Yes	No
Predictive result/Classification	Yes	TP(True Positive) 47,070	FP(False Positive) 8,210
	No	FN(False Negative) 6,421	TN(True Negative) 139,075

by five matrix. When a C.M is used for a binary problem as illustrated in Table 2.1 each of the four boxes has a specific name that is useful to memorize where:

1. True Positive (TP): The number of cases the classifier correctly predicted (the predicted value is yes and the actual value is yes)

2. False Positive (FP): The number of cases the classifier incorrectly predicted (the predicted value is yes but the actual value is no)

3. True Negative (TN): The number of cases the classifier correctly predicted (the predicted value is no and the actual value is no)

4. False Negative (FN): The number of cases the classifier incorrectly predicted (the predicted value is no but the actual value is yes)

2.4.3.1 Metrics Computed from a Confusion Matrix

Text in bold are the evaluation metrics that we can compute from C.M which tell easily whether the classifier is really appropriate or not.
Classification Accuracy (CA): Equation 2.5 overall, how often the classifier is correct.

$$CA = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.5}$$

Classification Error (CE): Also known as miss classification rate. We use Equation 2.6 to calculate CE

$$CE = \frac{FP + FN}{TP + TN + FP + FN} \tag{2.6}$$

Also we can calculate CE from Equation 2.7

$$CE = 1 - CA \tag{2.7}$$

Sensitivity: when the actual value is positive, how often the prediction is correct? How sensitive is the classifier capable of detecting positive instance?
Sensitivity is also known as "True Positive Rate (TPR)" or "Recall" and we can calculate it from a C.M as shown in Equation 2.8

$$TPR\ or\ Recall = \frac{TP}{TP + FN} \tag{2.8}$$

Specificity: When the actual value is negative, how often the prediction is correct? How specific or selective is the classifier in predicting positive instances? Specificity is something we want to maximize. Equation 2.9 is used to calculate it.

$$Specificity = \frac{TN}{TN + FP} \tag{2.9}$$

False Positive Rate (FPR): When the actual value is negative, how often the prediction is incorrect? The value of FPR is $1 - specificity$ and we can calculate it from Equation 2.10

$$FPR = \frac{FP}{TN + FP} \tag{2.10}$$

Precision: It answers the question, when a positive value is predicted, how often the prediction is correct? Precision describes how precise the classifier is when predicting a positive instance. We used Equation 2.11 to calculate it.

$$Precision = \frac{TP}{FP + TP} \tag{2.11}$$

F-Measure (F): Is defined as an harmonic mean of precision and recall. The standard F-measure is F1, which gives equal importance to recall and precision [24]. Equation 2.12 is used to calculate it.

$$F = \frac{2 \times precision \times recall}{precision + recall} \tag{2.12}$$

Kappa Statistic: It is used to measure the agreement between two sets of categorizations of a dataset while looking for chance agreements between the categories. It uses both the overall accuracy of the model and the accuracies within each category. Both terms use the predictive model and the actual sample points to look for chance agreement between categories [25]. The Kappa statistic varies from 0 to 1, where (see Equation 2.13)

$$Agreement\ is \begin{cases} No & \text{if } kappa = 0 \\ Slight & \text{if } 0.1 \leq kappa < 0.2 \\ Fair & \text{if } 0.21 \leq kappa < 0.40 \\ Moderate & \text{if } 0.41 \leq kappa < 0.60 \\ Substantial & \text{if } 0.61 \leq kappa < 0.80 \\ Near\ perfect & \text{if } 0.81 \leq kappa < 0.99 \\ Perfect & \text{if } kappa = 1 \end{cases} \tag{2.13}$$

Equation 2.14 is used to calculate Kappa statistic value

$$Kappa = \frac{Total\ Accuracy - Random\ Accuracy}{1 - Random\ Accuracy} \tag{2.14}$$

Where, Random Accuracy is calculated from Equation 2.15

$$Random\ Accuracy = \frac{(TN + FP)(TN + FN + (FN + TP)(FP + TP))}{(Total\ number\ of\ instances)^2} \tag{2.15}$$

We can conclude the evaluation metrics section in the following two points: First, C.M gives us a complete picture of how the classifier performs. Second, allows us to compute various classification metrics and how can these metrics guide our model selection process.

2.4.3.2 Receiver Operating Characteristic Curve (ROC)

The last steps we should care about, in the model building process, are adjusting the threshold taking into account [22]: First, threshold of 0.5 is used by default (for binary classifier) to convert predicted probabilities into class predictions. Second, threshold can be adjusted to increase sensitivity (reduced threshold) or specificity (increased threshold). Finally, sensitivity and specificity have an inverse relationship so, increasing one will decrease the other.

It seems incredibly inefficient to search for an optimal threshold by trying different threshold values to detect how sensitivity and specificity are affected by various thresholds. ROC curve gives us the ability to see how sensitivity and specificity are affected by various threshold without actually changing the threshold.

The ROC curve is a plot of the true positive rate on the y-axis against the false positive rate on the x-axis for all possible classification threshold as illustrated in Figure 2.7. It shows the trade-off between sensitivity and specificity. When the curve is closer to the upper left corner (0,1), the more accurate the classifier is. Otherwise, when the curve is closer to the random classifier of the ROC space, the less accurate the classifier is.

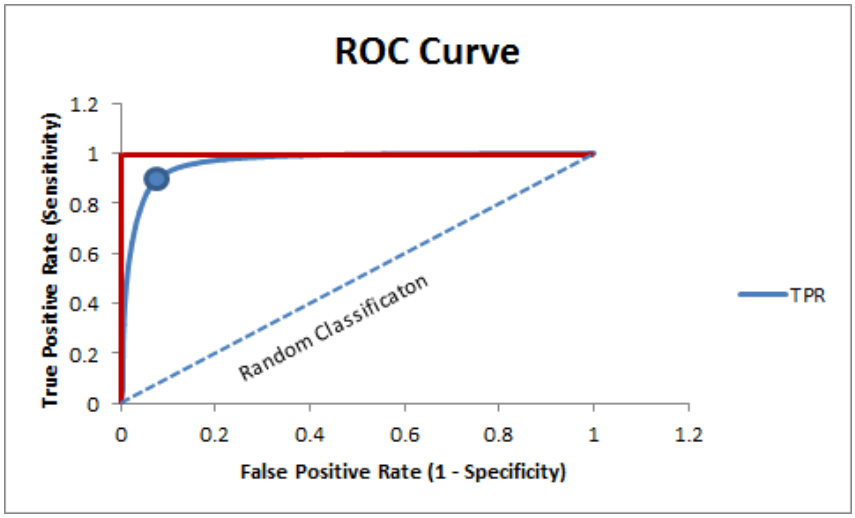

Figure 2.7: ROC Curve and AUC

2.4.3.3 Area Under the Curve (AUC)

AUC is another measure of classification accuracy. AUC is the area under the ROC curve (see Figure 2.7), meaning the percentage of the area that is located under the ROC curve. The closer the AUC to 1 the more accurate the classification is. AUC is a useful evaluation metric because it serves as a single number summary of classifier performance. If we randomly choose

one positive and one negative observation, AUC represents the likelihood that our classifier will assign a higher predicted probability to the positive observation. AUC is useful even when there is a high class imbalance (unlike classification accuracy) [26].

2.5 Semantic and Distance Matrices

There are many methods to calculate the similarity between two entities such as strings, numbers, rows of data, images and others [27]. Similarity is the measure of how much alike two data objects are [28]. Similarity measure in data mining context is a distance with dimensions representing features of the object [20]. High degree of similarity measure means small distance, while large distance will represent low degree of similarity. The degree of similarity is measured in the range of 0 to 1, Equation 2.16 explains the two main consideration measures.

$$Similarity = \begin{cases} 1, & if X = Y \\ 0, & if X \neq Y \end{cases} \tag{2.16}$$

Where X and Y are two objects.

Huynh,et al. [29] affirmed that popular string matching measures present the similarity of meta-data strings (author name, coauthors, affiliation, keywords) taking into account the author names based on family name and given name. These measures are: Edit Distance which is the distance between strings X and Y is the cost of the best sequence of edit operations that converts X to Y. Token-Based Measures convert the strings X and Y to token multi-sets and considers similarity metrics on these multi-sets. Hybrid Measures compare two long strings X and Y.

Naumann [28] summarized various similarity measures as illustrated in Figure 2.8. In the next subsections, we will introduce the characteristics of some popular similarity measures.

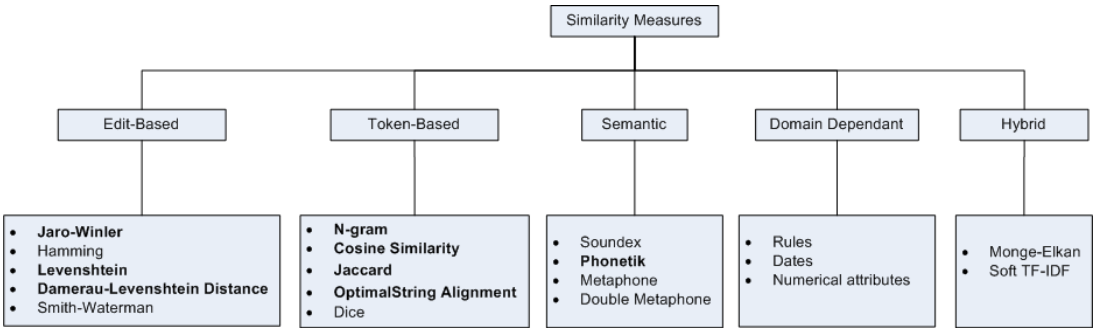

Figure 2.8: Similarity Measures, bold are the ones that we used

2.5.1 Semantics

Semantics is the discipline of deriving meaning from a collection of words or symbols [30]. In computing, it also has something to do with finding meaning in data and refers to a flexible way of modeling it. Computing sentence similarity is not a trivial task, due to the variability

of natural language expressions [31]. The method for finding the similarity between long text (documents) is counting the shared words between the text pair, but for the short text we should find the co-occurrence if founded. This is why finding the text similarity using the known algorithms is unfair such as, Jaccard, JaroWinkler, Levenshtein and others, especially with long text documents.

Before computing the semantic similarity, there are some basic preprocessing techniques including the following [31]:

1. Document preprocessing phase: It is important to normalize the documents and adjust them for our needs. We must obtain tokens from the text, especially if it was written by different authors from different backgrounds. This phase will transform the documents into a common form through the following steps:

   (a) Tokenization: Is the task of eliminating punctuation and other unwanted characters [32].

   (b) Stop words: Words that have a small impact in the matching process such as, (a, and, but, how, etc).

   (c) Stemming and lemmatization: Is the grammar and structure that documents use differently such as, *organize, organizes and organized*. Stemming is used to reduce all words to a common form such as three letters length, whereas, lemmatization removes the ending of the words and return the dictionary form, which helps to distinguish whether the word is used as a verb or as a noun. Both of them play an important roles in finding the semantic meaning behind the words which positively influence the semantic matching [33].

2. Dictionary-based measures: In order to capture the semantic similarity between two sentences, we need a dictionary-based measures such as:

   (a) WordNet: Is a large lexical database of English words, where nouns, verbs, adverbs and adjectives are organized by semantic relation to represent one concept [34].

   (b) DISCO: Is a java application with a multi languages database of distributional similar words called **word space**. It also gives us the ability to create our own database of similar words [35]. The word space of DISCO consists of two types:

      i. Col word space, which contains the word vector that represent the semantic meaning only (exclude the most similar words).
      e.g Col(house, home)=0.652

      ii. Sim word space, which represents the word vector and the most similar words for each
      e.g Sim(house,home)=0.7
      but Sim(gasoline, lemonada)=0.159

(c) Europarl Corpus: Is a collection of about thirty million words for eleven languages of European Union [36].

3. Transform Text to Vector: Transforming raw text into vector is a mandatory step for semantic algorithms [37]. Each vector has a length of vocabulary and each value in the vector is the number of co-occurrences of word to its index.

4. Once we formed the semantic vectors for the work titles pair, we computed the similarity between them using a mathematical model such as cosine similarity.

2.5.2 Levenshtein Distance

The notation of string changing operations is the fundamental of the Levenshtein distance idea in order to determine the degree in which two strings differ from each other. To transform string s1 into string s2 we need a minimum number of character insertions, deletions, and replacements [38]. For example Figure 2.9 describes the steps for finding the minimum edit distance between two strings (man and moon) which is $lev(man, moon) = 2$.

j / i	Null	m	o	o	n
Null	0	1	2	3	4
m	1	0	1	2	3
A	2	1	1	2	3
n	3	2	2	2	2

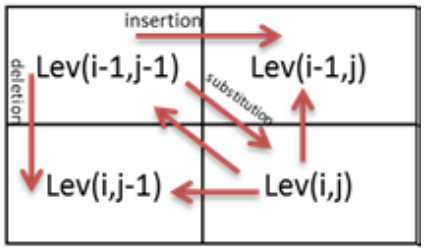

Figure 2.9: Levenshtein distance

Assume there are two strings, S_1 and S_2, the logic to calculate the Levenshtein distance $L_{S_1,S_2}(i,j)$ is shown in Equation 2.17 and 2.18 [39].
If $min(i,j) = 0$

$$L_{S1,S2}(i,j) = max(i,j) \tag{2.17}$$

Otherwise,

$$min \begin{cases} Lev_{S_1,S_2}(i-1,j)+1 \\ Lev_{S_1,S_2}(i,j-1)+1 \\ Lev_{S_1,S_2}(i-1,j-1)+1(S_{1i} \neq S_{2j}) \end{cases} \tag{2.18}$$

Where, $L_{S_1,S_2}(i,j)$ is the distance between the first i characters of S_1 and the first j characters of S_2.

2.5.3 Damerau Levenshtein Distance

Damerau Levenshtein distance is calculated in the same way as the Levenshtein distance is, except that transposition of two adjacent characters is also allowed as an edit operation [40]. For example, the Levenshtein distance between the words ppu and pup would be 2, but the Damerau Levenshtein distance would only be 1 since the transposition of p and u is considered a single operation. Damerau Levenshtein distance was mostly used for spelling checks as the four edit operations covers over 80% of human misspellings [41]. Damerau Levenshtein equation is similar to Levenshtein distance with one addition recursion for the transposition operations as illustrated in Equation 2.19

$$min \begin{cases} Lev_{S_1,S_2}(i-1,j)+1 \\ Lev_{S_1,S_2}(i,j-1)+1 \\ Lev_{S_1,S_2}(i-1,j-1)+1(S_{1i} \neq S_{2j}) \\ Lev_{S_1,S_2}(i-2,j-2)+1 \end{cases} \quad \text{if } i,j > 1 \; and \; a_i = b_{j-1} \; and \; a_{i-1} = b_j \tag{2.19}$$

2.5.4 Jaro Similarity

"Jaro is based on the number and order of the common characters between two strings; it takes into account typical spelling deviations and mainly used in the area of record linkage" [28, 42]. Jaro Winkler is an extension of Jaro distance which gives the first few letters more favorable rating and is used with longer string records [42]. If m is the number of matching characters and t is the number of transpositions then Equation 2.20 will determine the search range for matching characters and Equation 2.21 will calculate Jaro similarity between two strings x and y of length $|x|$ and $|y|$ respectively.

$$m = \frac{max(|x|,|y|)}{2} - 1 \tag{2.20}$$

$$sim_{jaro} = \frac{1}{3}\left(\frac{m}{|x|} + \frac{m}{|y|} + \frac{m-t}{|m|}\right) \tag{2.21}$$

2.5.5 Jaccard Similarity

Jaccard similarity (j) is computed as the number of shared terms over the number of all unique terms in both strings [42]. It can be used to represent the similarity between two documents where value is between 0 and 1. The value 0 means the documents are completely dissimilar while value 1 means the documents are full matched.

The Equation 2.22 is used to calculate the Jaccard Index in steps: First, it counts the number of unique words(union) in both sets. Second, it counts the total number of shared words(intersection)

in both sets. Finally, it divides the intersection number over the union number.

$$j(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \qquad 0 \leqslant j(A,B) \leqslant 1 \tag{2.22}$$

Where, A and B are two sample sets of length $|A|$ and $|B|$ respectively.

2.5.6 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them [42]. This technique makes sense in space that has multi dimensions with points represented as vectors with integers or boolean. The cosine distance between two points is the angle that the vectors to those points make. This distance is based on the relative frequency of words in a document.

The range of the angle is between 0 and 2π. $0°$ represents full similarity documents and above $90°$ represents dissimilar documents. Two vectors with the same orientation have a cosine similarity of 1, two vectors at $90°$ have a similarity of 0, and two vectors diametrically opposed have a similarity of -1 independent of their magnitude [43, 44].

Figure 2.10 shows how to calculate cosine similarity between the words man and $moon$ in the documents $Doc1$ and $Doc2$. For each document in the Cartesian coordinate system, there is a point that represents the count number of each of the words man and $moon$ in each of the two documents. $Doc1$, has 12 man words, and 14 $moon$ words , a point at $(12, 14)$. With regards to $Doc1$, the two words in $Doc2$ are represented with point $(5, 11)$. $Doc1$ document would be an arrow going from the origin to point $(12, 14)$ and $Doc2$ document would be an arrow from the origin to the point $(5, 11)$ and θ is the angle between the two vectors. Equation 2.23 is used to calculate the cosine similarity.

$$Similarity = cos(\theta) = \frac{A.B}{\|A\| \, \|B\|} \tag{2.23}$$

Where A and B are two vectors with length $\|A\|$, $\|B\|$ respectively.

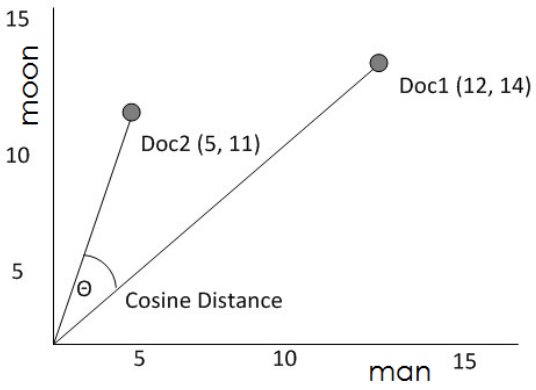

Figure 2.10: Cosine distance

2.5.7 N-Grams Similarity

N-grams are simply all combinations of contiguous words or letters of length n that can be find in the source text. Unigram refers to n-gram of size 1, Bigram refers to n-gram of size 2, Trigram refers to n-gram of size 3. Higher n-gram refers to four-gram, five-gram, and so on [45]. Figure 2.11 illustrates a typical example of a sentence may be considered as "This computer is a good one". Its unigram is considered a single word where its bigram considered as a pair of words and its trigram consists of three words.

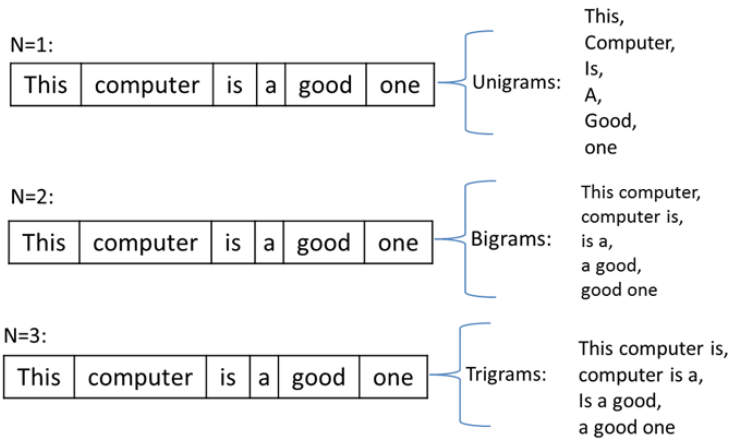

Figure 2.11: N-Grams overlapping

Equation 2.24 is used to calculate the distance between two strings a cording to the N-grams technique for each string.

$$|G(s1)| + |G(s2)| - 2\,|G(s1) \cap G(s2)| \tag{2.24}$$

Where, $|G(s1)|$ and $|G(s2)|$ are the combinations G of contiguous letters of length n for strings $s1$ and $s2$ respectively. The following steps are used to find the similarity based N-grams of size 3 between the two words $honorable$ and $honest$:

1. $G(honorable)$=$\{hon, ono, nor, ora, rab, abl, ble\}$=7
2. $G(honest)$=$\{hon, one, nes, est\}$=4
3. $G(honorable) \cap G(honest)$=$\{hon\}$=1
4. distance($honorable$, $honest$)=7+4-2*1=9

2.5.8 Optimal String Alignment

Optimal String Alignment Distance algorithm (OSA) edits the distance by changing characters if there are sentences that look like the other text. Then, according to OSA the "similarity" is

found and added to its total [46]. OSA is useful to fix spelling mistakes, editing mistakes and regular and irregular verbs. OSA works basically like Damerau Levenshtein algorithm (DL), the only difference is in how they transpose characters. DL algorithm can handle edits where the word has been misspelled twice while the OSA algorithm accepts one transposition and then moves on to the next substring.
For example, if we have two string: ot and two, then
$DL(ot, two) = ot \rightarrow to \rightarrow two = 2$
$OSA(ot, two) = ot \rightarrow t \rightarrow tw \rightarrow two = 3$

Chapter 3

Literature Review

Digital libraries and various meta-data schemes have rapidly grown. The internet and meta-data encoding in general move towards exchanges between a range of researchers, or similar research areas from several different authors, or even between different versions of the same scientific field [47]. This sheds light on the problem of name disambiguation. Databases and search options must be able to answer whether two articles were written by the same author or not. On the other hand, a researcher needs to know who exactly wrote this article for a purpose of future research collaboration or asking some questions about the data.

Many researchers have presented several definitions to illustrate the concept of AD. The author's name ambiguity decreases the performance, quality and reliability of information from digital libraries [9, 10, 48]. While [3, 49] defines AD as an author information that negatively influence the retrieval of bibliographic query results. In [29,50] AD integrates the bibliographical data of publications from heterogeneous sources into unified database.

During our review we will explain many procedures that have been used to introduce a solution for the problem of AD. Next sections will explain the most representative automatic AD methods found in the literature.

3.1 Type of Approach

The way to organize the author's names disambiguation methods is according to the type of approach they trade on. We will explain this in the following discussion [51]

3.1.1 Author Grouping Methods

This method uses clustering technique to find the similarity between the attributes from a group or references, or extracts the relationships among authors and coauthors from citations [10]. Different similarity functions such as, Edit Distance, Token-Based Measures and Hybrid Measures achieve the goal of finding the similarities between references for the same authors [29]. Also, there are many different ways to measure the similarity between all attributes of two

publications such as, Jaccard, Levenshtein, Jaro, etc [48]. Graph-based similarity method can create a co-authorship graph for ambiguous groups [51]. Each element of author's name and coauthor's name attribute is represented by a vector [10, 49, 50].

3.1.2 Unsupervised Techniques

Unsupervised learning clustering is the process of grouping a set of objects into classes of similar objects. In these approaches, ambiguous citations are clustered into groups of distinct authors by measuring the similarities between the attributes in the citations [3]. In this research, the most used techniques are [51]:

First, spectral clustering, clusters data that are connected but not necessarily clustered. It is mainly computed through eigenvalues and eigenvectors. The name disambiguation problem for author's citations is represented as a partition of the graph. This means that similar citations are tied to the same cluster.

Second, partitioning clustering, which constructs a partition of N documents into a set of K clusters. The final technique is the hierarchical clustering which groups the references of authors by building a tree-based Hierarchical taxonomy [3, 10].

3.1.3 Supervised Techniques

These approaches try to model all authors' patterns from a set of training data. In general, data provide insights in how to capture the implicit knowledge of a domain, and this method can be accurate and reliable when the data are represented well. Table 3.1 illustrates this.
Examples of supervised approaches are: Naive Bayes model [9], Support Vector Machine (SVM) [9, 29], Grouping algorithm [3], Deep Neural Network (DNN) [48, 52] and other approaches such as Random Forest, K-Nearest Neighbors (kNN) and Decision Tree (C4.5) [29].

3.2 Database

Various types of data with different format have been used in literature such as publication lists collected from the trusted and official web-sites e.g Microsoft Academic Search, ACM Digital Library, IEEE Digital Library, DBLP, PubMed, etc. Preparing and understanding the data is the first step to disambiguate the authors' names problem. Most of the proposed algorithms use the last-name and first-initial to refer to the same author [3, 9, 29, 48, 52] .

3.3 Features Information

Choosing the right features set is the most and the highest important step in the prepossessing phase in AD. Good features are those that can improve the accuracy and have the capability

to work with any new database. Here are some descriptions of some terms that are linked to features in this literature.

To disambiguate author's names, different features and methods have been used. Table 3.1 shows the information of the feature attributes in the literature. These kinds of evidences fall into three categories [51]:

The first category is Citation Information. It refers to the attributes that have been extracted from the citations such as coauthors' names, work title, journal title, year of publishing, etc. This citation information is commonly founded in all citations [3, 9, 10, 29, 47–50, 52, 53]. The category contains also some additional features such as emails, addresses, paper headers which are not always available. They usually help us to disambiguate names.

The second category is Web Information: It represents data from web used as additional information to improve the disambiguation task [50, 53]. On the other hand, there is one problem which is the additional cost of extracting all the needed information from the Web documents [51]. Web Information involves some steps for obtaining information [54]. It extracts a citation attributes, then it submits the attributes as a query to a search engine to find pages containing publications of the authors e.g. Query: "author's names" + "publications" + work title "Author's Names Disambiguation" and finally it collects the answer of each query.

The third category is Implicit Evidence which is divided into: First, author-related features such as email and author's name. Second, article-related features such as, article title, publication title, keywords, abstract , etc. which help a lot to estimate and calculate the similarities among references of authors.

Table 3.1: Outline of Disambiguation Procedures

Ref.	**Type of Approach**	**Used approaches**	**Used database**	**Evaluation The experiment**	**Used Features**
[9]	Supervised	Naive Bayes probability model Support Vector Machines	Publication from homepages DBLP citation databases.	Accuracy.	Co-author names The title of the paper The title of the journal
[3]	Supervised	Grouping Algorithm C-SVC binary classifier Cluster Filter	DBLP citation databases	Precision rate Recall rate Accuracy	Co-author The attribute title Venue
[10]	Unsupervised	K-way spectral clustering	DBLP citation databases Publication lists	Pair-Wise Similarity Metrics(CSM MSF) Similarity Metrics for Topic Correlation(TSM) Similarity Metrics for Web Correlation(MNDF)	Co-author Paper title Venue title
[48]	Supervised	Deep neural network string-matching measures(Jaccard Levenshtein, Jaro Jaro-Winkler, Smith-Waterman and Mogne-Elkan)	Vietnamese author dataset	k-fold cross-validation Accuracy	Author name Affliations Co-authors Paper keyword
[52]	Supervised (pairwise classification)	Deep neural network(DNN) DNN symmetry	DBLP databases	Precision (P), Recall (R) AUC and Accuracy	Co-authors names Paper title Publication venues
[49]	Unsupervised (name string only)	Author Co-citation Analysis (ACA)	PubMed	Factor analysis ACA mapping between lastname first initial and SPSS Oblimin	Chinese and Korean authors Last name, first initial
[29]	Supervised	Random Forest Support Vector Machine (SVM) Nearest Neighbors (kNN) C4.5 (Decision Tree) and Bayes	Microsoft Academic Search ACM Digital Library IEEE Digital Library	String Matching Measures(Edit Distance Token-Based Measures Hybrid Measures) Average accuracy	Author name Co-authors Affiliation Keywords in publications
[53]	Supervised (name string only)	Characteristic Scores and Scales (CSS)	Researcher-ID data of 4.271 registered researchers	A Spearman rank correlation Mean Normalized citation Rate (NMCR) Relative Citation rate (RCR)	Researcher ID At least 20 registered publication Year Chosen reference standard
[50]	Unsupervised	graph-based approach	Citeseer Digital Libraries	By-hand checking accuracy	Self-citation Co-authorship and Document source analysis

3.4 Experiments Evaluation

One of the most important point is to understand the evaluation of the experiments that have been used in the literature, to walk-through the various techniques used by the researchers to disambiguate authors' names problem. We should pinpoint on the following points:

3.4.1 The Mean and Standard Deviation

In [9, 10], the Mean and Standard Deviation statistical analysis were used to examine the accuracy ("according to ISO 5725-1 the general term accuracy is used to describe the closeness of a measurement to the true value") of experiments that have been used by some machine learning approaches to disambiguate authors' names in the paper title, journal title and the probability in which two citations belong to the same author. Therefore, using these statistical measurements show that changing the training data size will change the performance of the name disambiguation algorithm.

3.4.2 Experiments Design and Evaluation

Experiment design is a careful balancing of several features including, author name pattern, dataset, samples size, attributes and citations records. Every aspect in the designing phase plays an important role to construct a reliable model that can be obtained and correctly interpreted. Evaluating the experiment methods after they have been carried out is one of the core stages in the data process. Table 3.2 illustrates the setup and the evaluations techniques that have been used by different studies.

3.5 Related works

Unlike supervised machine learning approaches, Ibrahim M.Qdemat [55] proposed a heuristic based system in which all publications of a given author can be aggregated in the same profile.Qdemat also designed an algorithm that uses the publication meta-data and mainly he used email, name string similarity, affiliation and co-authors features.We will proposed a supervised machine learning approach that uses the features of ORCID citation attributes, to train the system to disambiguate author names.Particularly, to built classifier based on those attributes is a significant challenge.

In Shahd's Zeiad Ewawi [56] thesis, a comparison between different types of clustering algorithms was held to solve the ambiguity of author's names. Various features are involved in the clustering process: last name, author's other names, affiliation, title, abstract, keywords and co-authors. The aim of Shahd's study is controlling the behavior of the clustering algorithm in order to reduce the effect of the data uncertainty by applying a set of rules. Term Frequency

Inverse Document Frequency (TF-IDF) algorithm was used for extracting distinctive terms from features values.

Table 3.2: Outline of Disambiguation Performance

Ref.	Author name pattern	Data set	Data set size	Citation size	Citation based	Attributes	Experiment evaluation	Accuracy	Most useful Information
[3]	First name initial and Last name	DBLP website	476 individual authors	8,441	Yes	Coauthors Title Venue	Precision rate Recall rate	75%	Coauthors the most useful Title better than venue
[10]	First name initial and Last name	DBLP website	14 DBLP name data sets	400,000	Yes	Coauthors Title Venue	Standard Deviation	61.5% to 64.7% average accuracy	Coauthors the most useful Title better than venue
[53]	Author profile	Researcher-ID data from 8 selected countries	20 different authors	4,271	No	Publication year Researcher ID Field of study	Relative Citation Rate (RCR) Mean Normalized Citation Rate (NMCR)	69.80%	Field of study Publication year
[9]	First name initial and Last name	DBLP website	9 individual authors	300,000	Yes	Author names Paper title Journal title	The mean and The standard deviation (StdDev)	Two data sets achieved 90% by SVM Average=73.3% by Naive Bayes approach Average=65.4%,by SVM	Journal title
[48]	Author instance names	Online digital libraries: IEEE Xplore ACM MAS	10 data sets	30,537	No	Author name Co-author Affiliation keyword	Prediction error rate F-measure	91.31%	Features not ranked
[29]	Family name and given name	Online digital Libraries: Microsoft Academic Search ACM Digital Library IEEE Digital Library	10 data sets	4,350	No	Coauthor names Paper title keywords Journal title keywords	Non	98.33%	Features not ranked
[50]	Family name	Citeseer	8 data sets	4,799	Yes	Co-authorship and Source URL meta-data	Precision Recall F-measure	Achieving precision of 0.997 Recall of 0.818	Features not ranked
[5]	Author instance names	Scopus and WoS	Undefined	1,518	Yes	Coauthors Paper title URLs of the papers The name of the scholar	Precision Recall F-measure	Precision=95.38% Recall=96.24% F-measure=94.54%	Features not ranked

Chapter 4

Methodology and Data Analyses

In this chapter we provide the comprehensive and detailed analysis of data sets for AD. The main result of existing data sets is created, represented, and used. In order to present a solution for AD, we must use a representative database that covers various names from different nationalities. In this section we will describe the technique and the process to build the data for validation stage.

4.1 ORCID Database

Once per year, ORCID releases a downloadable data file that contains a copy of all public data in its registry. This file contains the public information associated with each ORCID user in both JOSN and XML format. In this thesis, we used Orcid Public-Data-File-2015 which contains over 1.6 million records.

4.1.1 Datasets Selection

Selecting the datasets from ORCID database is the most important phase. Our criterion is to have variety samples that would represent as much cases as possible in the entire database. The selection methods are based on the following steps:

1. We determined the first name and last name from ORCID profile files for each author.

2. LNFI is used as a primary key to filter the authorship records of ORCID datasets. We retrieved from ORCID database all author strings of the selected LNFI . Then, all citations which consist of coauthors name strings, journal title, work title and the year of publishing that contains one of these author names are retrieved from ORCID database.

3. We calculate the frequency of the distinct LNFI then we retrieve all documents based on these names from ORCID database. Calculating the frequency for each LNFI helps us in choosing different sizes of representative authors' clusters. This is a good indication for evaluating the disambiguation of names.

4. We select different LNFI from various backgrounds to have better representations of all authors.

4.1.2 Database and Labeling

This section describes the data sets that we use for the experimental and evaluation phases. Table 4.1 lists a sample data sets that we choose randomly in our experiments. These data sets are labeled from ORCID engine through the registration process.

Table 4.1: Orcid Datasets Samples

#	Dataset	Criteria (LNFI)	Size of documents	ACC Records
1	Wang	Wang,Y	916	86,462
2	Zhang	Zhang,Y	829	293,472
3	Lee	Lee,J	717	176,820
4	Li	Li,J	563	46,010
5	Kim	Kim,J	678	332,070
6	Ahmad	Ahmad,K	16	9,702
7	Liu	Liu,Y	672	78,462
8	Chen	Chen,Y	545	9,120
9	Park	Park,J	350	52,670
10	Kumar	Kumar,A	297	118,680
11	Wu	Wu,J	252	141,000
12	Silva	Silva,A	160	266,772
13	Brown	Brown,D	41	177,662
14	Islam	Islam,M	127	12,240
15	Sun	Sun,X	121	2,450
16	Ali	Ali,M	106	756
17	Carvalho	Carvlho,A	71	16,256
18	Fernandes	Fernandes,A	48	54,522
19	Nguyen	Nguyen,T	169	96,087
20	Santos	Santos,R	72	3,306
21	Smith	Smith,J	122	1,640
22	Wang	Wang,X	637	77,588
23	Zhang	Zhang,X	566	2,250
24	Baker	Baker,M	16	56
25	George	George,J	18	1,560
26	Liu	Liu,C	303	237,656
27	Oliveira	Oliveira,A	87	10,2080
28	Sharma	Sharma,S	167	2,162

We will use cross validation technique to assess the performance of machine learning model which enables us to know how the machine learning model would be generalized to an independent data set and to estimate how accurate the predictions will be in practice. Table 4.2 represents the total number of citation records that were chosen randomly to represent all data sets in Table 4.1.

We split these citation records into two types of data sets, the known data (Cross validation set) and the unknown data (Final test set). By using cross validation, we would be "testing" our machine learning model in the "training" phase to check for overfitting and to generalize the model we will use for an independent data, which is the testing data. For cross validation rounds,

we divided our original training data set into two parts, cross validation training set and cross validation testing set or validation set.

Table 4.2: Training and Testing samples

#	Test Purpose	Quantity	Total number of citations pairs
1	Cross validation set	1606214	803107
2	Final test set	401552	200776

4.2 Methodology

4.2.1 Design Rules

Our aim is to provide a sequence of different cases to guide us during the design process strategy in order to reach the goal of solving the names ambiguity depending on ORCID citations. ORCID represents authors from various backgrounds and cultures; this variety leads to different labels which are reported to be ambiguous in author's names. Towards the end of the following subsections, we explain the main keys of our design principles.

4.2.1.1 Notes on ORCID Profile

The following are some points that we should be aware of through the design phase process

1. ORCID-profile: Each author has one profile with only one ORCID-ID.
2. There are some blank elements in author's profiles in ORCID database.
3. Some elements have all the fields while others do not.
4. The main source of author's details is ORCID-activities which consist of affiliations and ORCID-work.
5. Affiliations include the department's name and the organization's name. However, affiliations are not included in some parts of ORCID profiles. At least one of them is listed for the author who has creative works.
6. ORCID-activities do not exist for an author who asks for an ORCID-ID also, ORCID-work does not exist for someone who is involved in a scholarly work.
7. Keywords are not always available.

In the next subsections we propose a method using ORCID open database to solve the AD problem based on citations records.

4.2.1.2 Citation Representation

In ORCID database, each profile that belongs to an active author has at least one citation record which consists of various information such as, co-authors, journal title, work title, publishing year,..etc. As shown in Figure 4.1, citations records have two different types, Bibtex formatting style and None-formatted style.

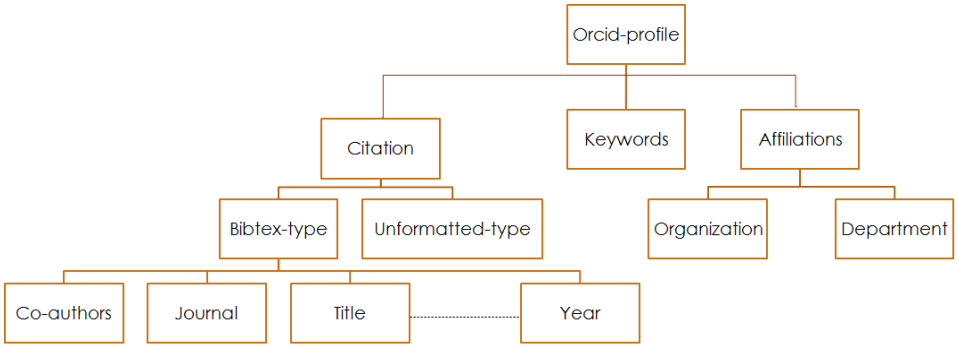

Figure 4.1: Citation Architecture

Our citation criteria will be based on the following points:

1. In the query result we will exclude the none-formatted citation type from the data sets.
2. The citation features based on co-authors, journal title, work title and publishing year.
3. Random number of citation records was represented in each data set sample.
4. In order to establish positive and negative pairs, number of citations in any profile must be more than 2.
5. Positive pair P_{pair} is the representation of any two different citations from the same author's profile, e.g

   0000-0002-3562-2323;**16051195**;Zhang,Y

   0000-0002-3562-2323;**14022797**;Zhang,Y

   The bold number in the ORCID ID is a primary key for citation records.

   To collect these pairs we used two methods:

   First, compiling pairs of citations from the same profile, so that these pairs are separated and not repeated in any record of the other pairs. Figure 4.2 illustrates this criteria. The number of positive pairs n from each dataset could be calculated according to Equation 4.1.

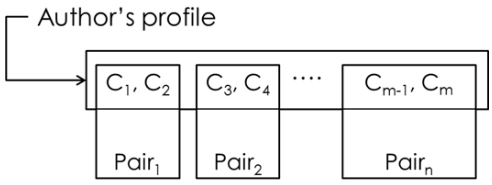

Figure 4.2: Positive pairs for separated citations from an author's profile

$$n = \sum_{First_p}^{Last_p} \left\lfloor \frac{m}{2} \right\rfloor \tag{4.1}$$

Where, $First_p, Last_p$ are the authors' profiles range in each dataset, and m is number of citations in author's profile. We use this method to generate more positive pairs. Algorithm (1) used to collect a random number of positive citation pairs from separated citations belongs to the same author's profile.

Algorithm 1 Positive Citation Pairs Algorithm from separated citations

```
1: Function P_pair(Profile p)
2:    for p in Dataset LNFI
3:       {
4:          let i=1
5:          let j=2
6:          for citation m in p (1 to random)
7:             {
8:             pair=(m[i],m[j]);
9:             return pair
10:            i=j+1
11:            j=i+1
12:            }
13:      }
14: Endfunction
```

Second, compiling chain of positive pairs of citations from the same profile. The first pair consists of the first two citations then the second citation is linked to the third one to construct the second pair and so on. Figure 4.3 illustrates this criteria. The number of pairs n from each dataset can be calculated according to Equation 4.2.

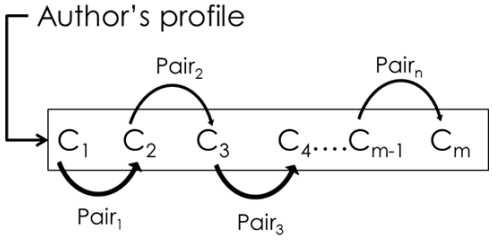

Figure 4.3: Positive pairs for one author profile (Chain of citations)

$$n = \sum_{First_p}^{Last_p} m - 1 \tag{4.2}$$

Algorithm (2) is used to collect a random number of positive pairs from chain of citations that belong to the same author's profile.

Algorithm 2 Positive Citation Pairs Algorithm

```
Function P_pair(Profile p)
for p in Dataset LNFI
    {
        let i=1
        let j=2
        for citation m in p (1 to random)
            {
            pair=(m[i],m[j]);
            return pair
            i++
            j++
            }
    }
Endfunction
```

6. Negative pairs are the representation of any two citations from different authors' profiles, e.g.

   0000-0001-7636-7368;**14180633**;Zhang,Y

   0000-0001-8286-300X;**15159439**;Zhang,Y

   To collect these pairs we used two methods:

   First, compiling pairs of data from different profiles so that each record from author's profile $P1$ is linked to one record from a different author's profile $P2$ without duplication. Figure 4.4 illustrates this criteria.

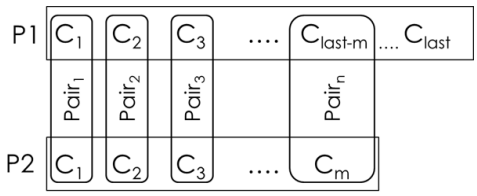

Figure 4.4: Negative pairs (First Method)

The number of pairs n from each dataset could be calculated according to Equation 4.3.

$$n = \sum_{i=1}^{\#\ profiles} min(\#citation_{p_i}, \#citation_{p_{i+1}}) \tag{4.3}$$

Where, $\#citation_{p_i}$ and $\#citation_{p_{i+1}}$ are the minimum number of citations in both profiles. Algorithm (3) is used to collect a random number of negative pairs form two different authors' profiles taking into account the one-to-one relationship between the citations in each profile.

Algorithm 3 Negative Citation Pairs Algorithm(one-to-one relationship)

```
1: //P1,P2 are two different profiles
2: let x=count(citation in P1)
3: let y=count(citation in P2)
4: let z=min(x,y)
5: Function GetCitation(Profile p, int i)
6:     {
7:         for citation c in p
8:         if c[i]==True //is found
9:         return c
10:        else()
11:        endif
12:    }
13: Endfunction
14: for i=1 to z
15:    {
16:        First=GetCitation(P1,i)
17:        Second=GetCitation(P2,i)
18:        return pair(First, Second)
19:    }
```

Second, compiling pairs of data from different profiles so that each record from profile $P1$ is linked to all records in profile $P2$ without duplication. Figure 4.5 illustrates this criteria.

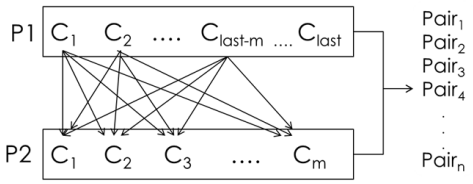

Figure 4.5: Negative pairs (Second Method)

The number of pairs n from each dataset could be calculated according to Equation 4.4

$$n = \sum_{i=1}^{\#\,profiles} \frac{\#citation_{p_i} \times (\#citation_{p_{i+1}} - 1)}{2} \tag{4.4}$$

Algorithm (4) is used to collect a random number of negative pairs from two different authors' profiles taking into account the one-to-many relationship between the citations in each profile.

Algorithm 4 Negative Citation Pairs Algorithm(one-to-many relationship)

```
1: //P1,P2 are two different profiles
2: let x=count(citation in P1)
3: let y=count(citation in P2)
4: Function GetCitation(Profile p, int i)
5:     {
6:         for citation c in p
7:         if c[i]==true //is found
8:         return c
9:         else()
10:        endif
11:    }
12: Endfunction
13: for i=1 to x
14:    {
15:        First=GetCitation(P1,i)
16: for j=1 to y
17:    {
18:        Second=GetCitation(P2,j)
19:        return pair(First, Second)
20:    }
21:    }
```

The next subsection describes the citation methodology based on Co-authors, Journal title, Work title and Publishing year features.

4.2.2 Coauthor Methodology

Since ORCID contains records from different backgrounds, we found various format of the author's names inside each citation block. We summarized them as follows:

1. One part string name. The family name or the first name.

   e.g

   kumar,

2. Two parts string name. The full first name followed by the full family name or the full family name followed by the full first name. The same for names with abbreviations.

   e.g

   kumar, akhilesh

   kumar, A.

3. Three parts string name which represents the first name, middle name and the family name. In some cases one or two of the names parts with their first initials

   e.g

   Alexander, Swathy A

Alexander Swathy Ann
Alexander, S. A.

4. Four parts string name which represents the first name, two middle names and the family name. In some cases from one to three names written with their first initials

   e.g

   Permana A D C
   Permana, Antonius Dimas Chandra

ORCID File Structure subsection clarifies the personal information for each author. The arrangements of the main authors' names are a sequence of the given-name (first name) then the family name. Figure 4.6 describes the representation of the coauthors in the bibtex citations format which comes in two types, last name then first name (or the abbreviation LNFI) or first name then family name (or the abbreviation FNLI).

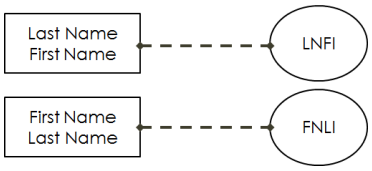

Figure 4.6: Coauthor's Names Structure

We implemented Jaccard, JaroWinkler, Cosine, Levenshtein and NGram string similarity technique to represent the similarity between two pairs of coauthors lists. In order to be more accurate, we believed that we had to take into account the ambiguity of coauthor's names, so we calculated the similarities in two ways:
Firstly, Full Matching(FM) names when the two coauthors are completely identical. For example, when the similarity score is 50% that means one coauthor's name is shared in two citations records. When the similarity score is 75% that means two coauthor's names are shared in two citations records. When the similarity score is 100% that means three or more coauthor's names are shared in two citations records.
Secondly, When the number of shared coauthor's names in two citation records is less than or equal to two. Then the algorithm will calculate the Partial Similarity(PS) match as the same technique of the full string match. The weight of the PS will be 25% and this will ensure that the partial match will not exceed 25% of the Total Similarity(TS) of any pair. Equation 4.5 is used to calculate the coauthor's pairs score.

$$TS = FM + (PS * 0.25) \tag{4.5}$$

Algorithm (5) introduces the way of calculating the TS between each positive or negative pair in coauthors' list.

Algorithm 5 Coauthors Score Algorithm

initialize i,j,Counter1,Counter2 to zero
$Function\ GetScore(Algorithm\ A, Coauthors\ c1, c2)$
    {
        score=A(c1,c2)
        $return\ score$
Endfunction
    }
$Function\ GetSimilarity(Counter\ C)$
    {

$$Match = \begin{cases} 0.5 & \text{if } C = 1 \\ 0.75 & \text{if } C = 2 \\ 1 & \text{if } C \geqslant = 3 \end{cases}$$

    $return\ Match$
    }
Endfunction
for Coauthors c1 in citation1
    {
        i++
        For Coauthors c2 in citation2
            {
                j++
                $score = GetScore(G_{name}, c1[i], c2[j])$

$$where,\ G_{name} = \begin{cases} \text{Jaccard} \\ \text{JaroWinkler} \\ \text{Cosine} \\ \text{Levenshtein} \\ \text{NGram} \end{cases}$$

                If score==1 then
                    Counter1++
                else
                    Counter2++
        }
if Counter1>=3
            {
        FM=GetSimilarity(Counter1)
    else
        FM=GetSimilarity(Counter1)
        PS=GetSimilarity(Counter2)
            }
        TS=FM+(PS*.25)
    }
$return\ TS$

This technique is designed to ensure that two similar items generate hashes that are themselves similar. In fact, the similarity of the hashes has a direct relationship to the similarity of the

citations they were generated from. The final step is to use a machine learning technique to choose the promising similarity method that helps us to predict whether each pair of coauthors belongs to the same author or not.

4.2.3 Publishing Year Methodology

We believe that publishing year plays an important role as the other used features. It could give some indications about authors, especially when they submit everything new in their field of research. We found a lot of researchers publishing in the same year or within a period of two or three years. The criteria of calculating the score of publishing year based on the following: First, The publication year score (Y_s) is equal to one if two publications were published in the same year. Second, If the publications were published in different years, then the Y_s will calculated as shown in Equation 4.6.

$$Y_s = \frac{1}{|y_2 - y_1|} \tag{4.6}$$

Where $|y_2 - y_1|$ is the absolute value of the difference between the publication years from the citation pair.
Algorithm (6) describes the criteria of calculating the score of any publication years.

Algorithm 6 Publishing year Score Algorithm

```
1: Function year(pair y2, y1)
2: {
3: D = |y2 − y1|
4:       if D==0 then
5:             Ys=1
6:             else
7:             Ys=1/D
8: return Ys
9: }
10: EndFunction
```

4.2.4 Journal Title Methodology

Authors publishing their researches in the same journal or in different journals are highly concerned about the correct journal to submit their papers. The question regarding journal choice is : should they submit their work to the same journal or not? Technically, it is perfectly acceptable to submit two or more papers to the same journal. This way ensures that the author's readers will get a clear idea about his work, making the formatting much easier and the author more familiar with the style of his work. Some authors believe that publishing in different journals shows that their work has been accepted by a wide range of academics and becomes

widely known [57]. Equation 4.7 calculates the similarity value between Journals.

$$J_s = \begin{cases} 1 & \text{if } j_a = j_b \\ 0 & \text{if } j_a \neq j_b \\ > 0 \; and \; < 1 & \text{if } PS(j_a, j_b) \end{cases} \tag{4.7}$$

Where J_a, J_b are the Journal titles that belongs to a pair of negative or positive citation and J_s is the Journal similarity score. The technique we followed to compute the similarity between each pair of journals consists of the following steps:

Step1: Extract the Journal feature from the citations records for each LNFI data set in ORCID database. In this step, we removed the stop words and clean the journal names from unwanted symbols.

Step2: In order to find the score of the similarity between any pair of journals, we implemented seven string algorithms: Jaccard, Jaro, Cosine, Levenshtein, NGram, Soren distance and Damerau Levenshtein Distance.

Step3: The final step is to use a machine learning technique to choose the promising similarity method that helps us to predict whether each pair of Journals belongs to the same author or not. Figure 4.7 describes the process of calculating journal similarity using each string algorithm.

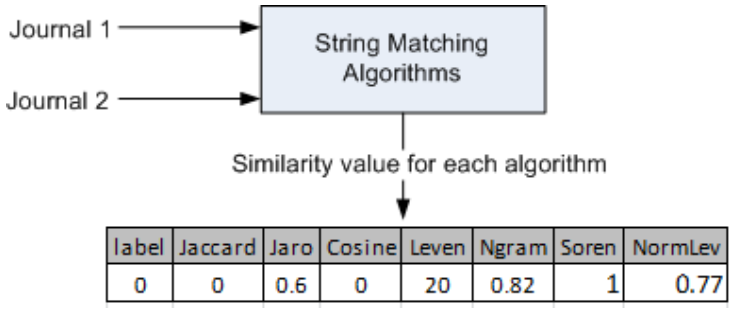

Figure 4.7: Journal Similarity

4.2.5 Work Title Methodology

Computing string similarity is not a trivial task, because of the different of natural language expressions [31]. The technique of finding similarity between long texts focuses on the shared words in both, but in short text it focuses on word co-occurrence. Computing authors' work titles using the traditional similarity measures is useless because these measures are based on the syntactic features and other based measures.

Figure 4.8 shows the functionality of finding the similarity between two work titles. In step 1 and 2, different techniques were used to find the semantic similarity between positive and negative work title pairs. We must obtain tokens from the work title and apply various normalization to adjust them for our needs. This will reduce the influence of grammatical reasons in addition to how words are formed, and their relationship to other words in the same language. We achieve that by: (1)Title tokenization; (2)Removing stop words; (3)Stemming and

lemmatization. Step 3 represents the total number of items in each title. Our methodology is able to find the semantic meaning of any title which consists of one word or more as clarified in step 4. Each title is treated as a bag of words or terms and is represented in a high dimensional space as a vector.

The dimension of the space depends on indexing terms which are chosen to be relevant for the collection dictionaries database. A collection of n documents can be represented in two vectors: Category vector represents the terms of the first sentence in the title pairs and the input vector represents the terms of the second sentence as shown in step 5. Each item in the vectors represents a dimension and all dimensions establish the space model. In step 6 we formed the semantic vector for each work title by processing each title sentence using a collection of dictionaries (En-Wikipedia, En-PubMedOA, En-BNC and WordNet) to help us extract the concepts related to the terms in each sentence. In step 7 the category vector consists of the terms and the occurrence of each term in the documents which has been retrieved from the dictionaries as illustrated in matrix $Category$ in Equation 4.8

$$Category = \begin{bmatrix} Term & Frequency \\ t_{21} & f_{22} \\ . & . \\ . & . \\ t_m & f_{mn} \end{bmatrix} \tag{4.8}$$

Where, m is the number of unique terms in all documents, each row represents the term and the occurrence in all documents.

The input vector in Equation 4.9 is the same as the category vector but it ignores the terms that are not related to the synonyms in the category vector.

$$Input = \begin{bmatrix} Term & Frequency \\ t_{21} & f_{22} \\ . & . \\ . & . \\ t_m & f_{mn} \end{bmatrix} \tag{4.9}$$

Once we formed the semantic vectors for the work titles pair in the space model, we computed the cosine similarity between them as shown is step 8 and finally step 9 represents the final result.

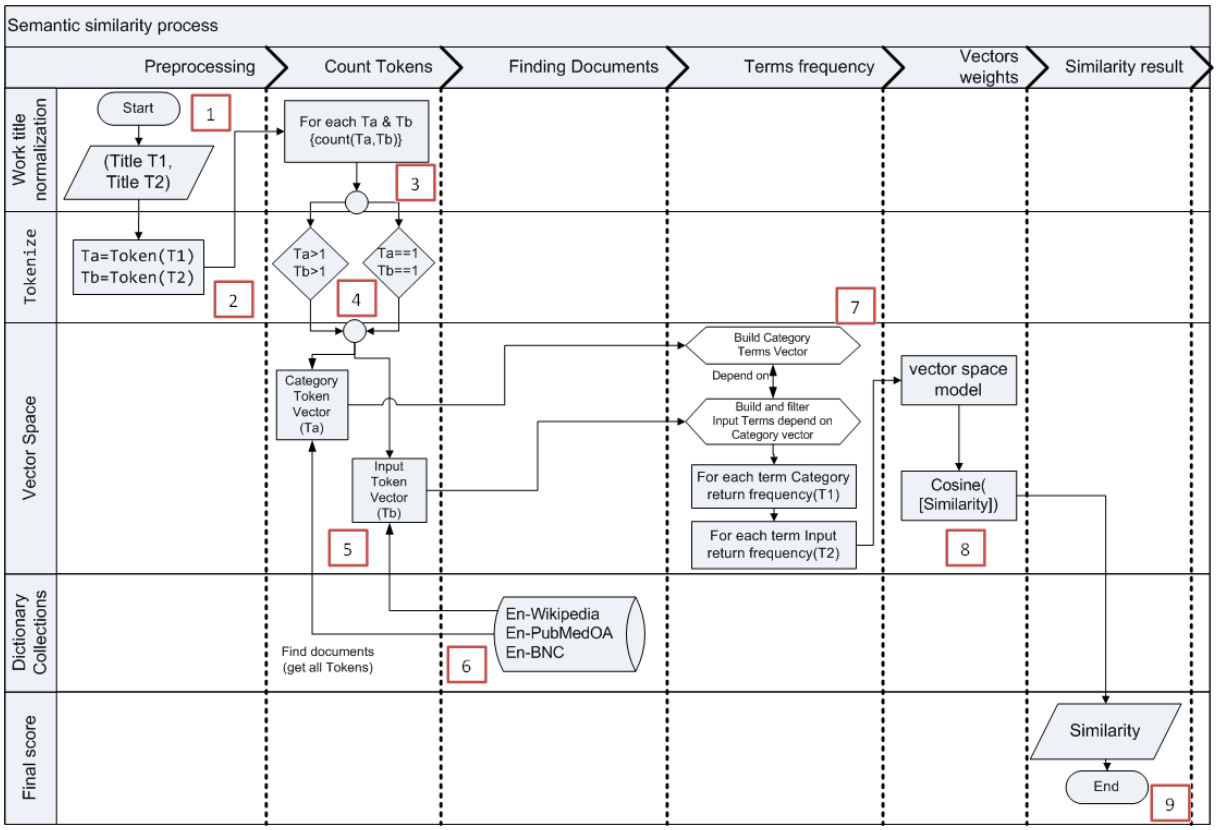

Figure 4.8: Sentence similarity computation diagram

4.2.6 Machine Learning

In this section, we will use the dataset that we derived in Table 4.2. The dataset is comprised of 1003,883 citation pairs. Our goal is to build a machine learning algorithm capable of detecting the correct citation (match or not match) in new unseen citation. In machine learning, this type of problems is called classification. Figure 4.9 illustrates the machine Learning phases. Training phase: In this phase, we train a machine learning algorithm using a dataset comprised of the citations and their corresponding labels. Prediction phase: In this phase, we utilized the trained model to predict labels of unseen citations.

The training phase for Author's Names Disambiguation (AD) classification problem has two main steps: First, Feature Extraction: In this phase, we utilized domain knowledge to extract new features that will be used by the machine learning algorithm. Coauthors' names, journal title, work title and publishing year are the features used in AD classification. Second, Model Training: In this phase, we utilized a clean dataset composed of the citations features and the corresponding labels to train the machine learning model. In the prediction phase, we applied the same feature extraction process to the new citation and we passed the features to the trained machine learning algorithm to predict the label.

In the ORCID dataset, we classify each record into one of two classes: First, class Yes means the citation pairs are matched and belongs to the same author. Second, class No means the citation pairs are mismatched and belongs to two different authors. Figure 4.10 illustrates the

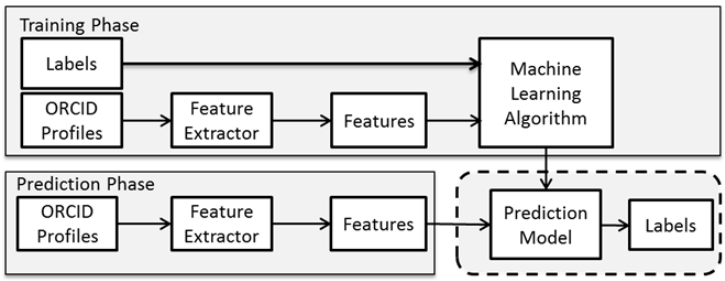

Figure 4.9: Machine Learning Phases

classification process which uses the training set to train the model to predict the unseen data (Testing set) using various models (Decision Tree (C4.5), Random Forest, Deep Learning and Naive Bayes).

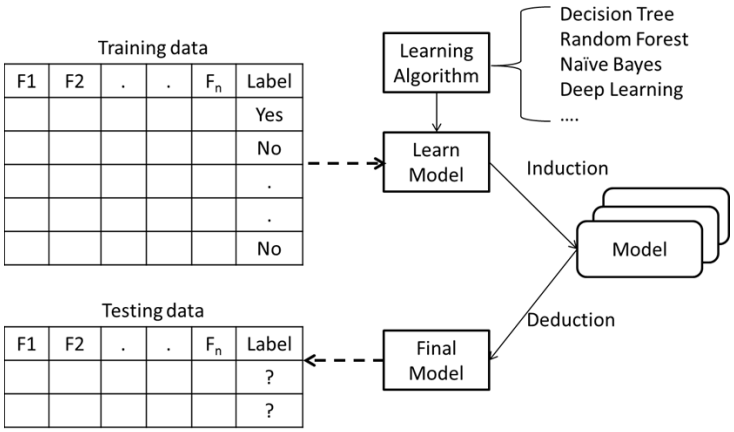

Figure 4.10: Weka Classification

In order to build a promising model that can achieve a high accuracy and a precise prediction on a set of unseen data we used various classifiers. Decision Tree, since the dataset contains continuous attributes, we utilized $C4.5$ as the primary algorithm which represented as J48 in Weka. In Random Forest there are only a few hyper-parameters we need to tune in Weka: The trees to build in Random Forest, the tree depth and the number of features which should be used for each tree. Using Naive Bayes, we assumed that the value of features are independent from others and that features have equal importance. Deep Learning which is represented as deeplearning4j in Weka. It is possible in Deep Learning to load predefined architectures as neural network and train it on a new dataset. In order to get additional performance we tweaked some parameters such as number of layers, learning rates, convolutional filters and processing parameters.

4.3 Chapter Summary

In this chapter as illustrated in Figure 4.5, we presented the methods we used for extracting the features of the database, as well as the methodology that have been used to calculate the

similarity between the negative and positive citation pairs. A matrix of real numbers has been obtained from records scores that represent the similarity of each citation pairs in the data-sets consists of coauthors, journal title, work title and publishing year. Different machine learning classification approaches used to build a promising model that can achieve a high accuracy and a precise prediction for unseen citation records.

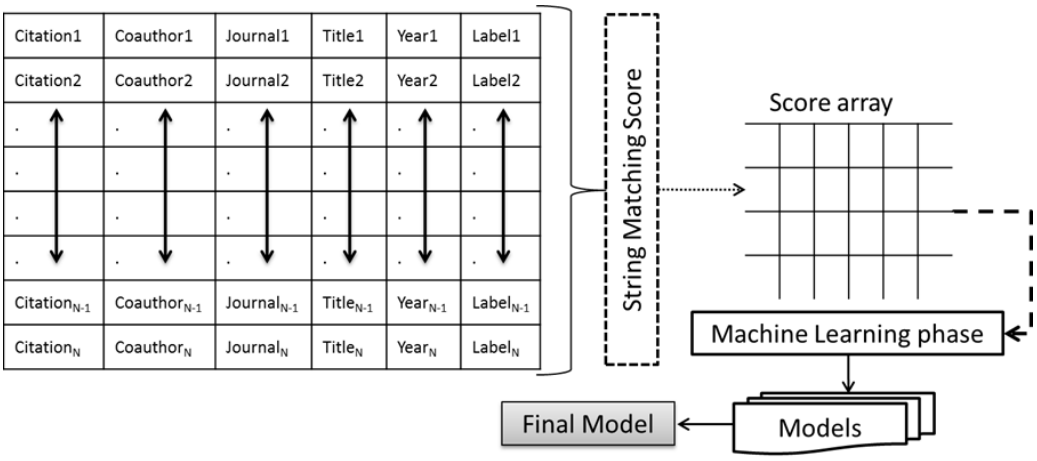

Figure 4.11: The Methodology Phases

Chapter 5

Experiments and Results

In this chapter, different experiments are conducted in order to evaluate the improvement brought out by our methodology approach. The first section will describe the experiments setup. The evaluation approach of the classification result will be described in section 2. The comparison between Random Forest (RF) versus Naive Bayes Classifier, Deep Neural Networks (DNN) and Decision Tree (J48) will be described in section 3. The last section will illustrate and analyze the experiment results, ranking the features, ranking the distance matrices and the execution time of predicting new instances.

5.1 Experiments Setup

The classification methods were tested by using 10-fold cross validation technique to compare the performance of the machine learning approaches that we used. In each round of cross validation we partitioned the original data sets into training set and testing set. We trained our machine learning model using the cross validation training set and test it against the testing set. The training set contains 803,107 instances and the testing set contains 200,776 instances. We calculated the Area Under Curve (AUC) of the machine learning models by validating the predicted results against the test set. Averaging the accuracies derived in all the 10-folds cases of cross validation to estimate the accuracy of our model.

We implemented four machine learning approaches, namely RF, Naive Bayes Classifier, DNN and J48 which were intensively tested in a comparative approach. RF has different parameters to be tuned, F_{num}, T_{num} and T_{depth}. Increasing F_{num} parameter shows how many attributes are selected to build one of those trees in RF. In general it improves the performance of the model. T_{num} refers to how many trees the forest is composed of, higher number of trees gives strong predictions. T_{depth} indicates the depth of each generated tree.

Table 5.1 summarizes the accuracy results according to RF tuned parameters. The accuracy value is directly proportional to the number of trees $T_{num} = 300$ in RF with the default value of $T_{depth} = 0$ and $F_{num} = 4$. After 300 trees there is no significant improvement. There is no need to go for pruning since the data samples expanded into each individual tree have already gone

through bagging.

Table 5.1: RF tuned parameters using 10-folds cross validation

#Experiment	T_{depth}	F_{num}	T_{num}	Accuracy
1	3	4	10	84.05
2	3	4	20	84.90
3	3	5	20	85.03
4	4	15	20	85.40
5	4	5	20	85.70
6	4	6	20	85.70
7	4	8	20	85.70
8	4	9	20	85.80
9	4	10	20	85.80
10	4	8	40	85.81
11	4	9	300	85.88
12	4	10	30	85.90
13	6	8	40	86.80
14	10	8	40	89.64
15	12	4	144	90.10
16	12	15	65	90.50
17	12	8	65	90.65
18	14	4	400	91.12
19	15	4	144	91.51
20	18	5	50	93.17
21	20	7	10	94.05
22	30	4	300	94.40
23	25	4	60	95.62
24	2	4	1,000	83.84
25	4	4	300	85.88
26	0	4	10	94.02
27	0	4	20	94.36
28	0	4	50	94.63
29	0	4	100	94.71
30	**0**	**4**	**300**	**94.77**

Next, we evaluated the effects of feature selection on classification performance using group of performance measures: Classifiers Accuracy (ACC), Precision, Recall, F-measure, ROC graphs and AUC. Performance evaluations were compared against each other in terms of classification results.

5.2 Random Forest Classifier

In order to validate the performance of RF classifier, we should be certain that the classifier has been well trained thereafter it can be evaluated using the testing set to figure out its accuracy. The following subsections explain the validation process.

5.2.1 RF statistic Result

We calculated the accuracy of RF Classifier model by validating the predicted results against the testing set. Table 5.2 summarizes the evaluation result considering the parameters $F_{num} = 4$, $T_{num} = 300$ and $T_{depth} = 0$. Kappa statistic measurement is equal to 0.87 which signifies near perfect agreement to predict the positive citation (class yes). Table 5.3 shows the detailed accuracy result by class in the testing phase. 93% of the citation pairs belong to the same authors and 96% of citation pairs belong to different authors and were correctly classified according to precision measurement.

Whereas a recall of 88% citation pairs is labeled as belonging to same authors and 97% citation pairs are labeled as belonging to different authors. Usually, precision and recall scores are not discussed in isolation. F-Measure makes the balance between precision and recall of 90% and 96% for class "yes" and "no" respectively. The Precision-Recall (PRC) area of 97% gives us a strong estimation that RF was successful to predict the citations that belong to the same author.

Table 5.2: The statistics results in the testing phase

#	Test	Result
1	Correctly Classified Instances	94.78%
2	Incorrectly Classified Instances	5.22%
3	Kappa statistic	0.87
4	Mean absolute error	0.09
5	Root mean squared error	0.20
6	Total Number of Instances	200,776

Table 5.3: RF detailed accuracy results in testing phase

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	PRC Area
no	0.97	0.12	0.96	0.97	0.96	0.98	0.99
yes	0.88	0.03	0.93	0.88	0.90	0.98	0.97

ROC curves represent excellent, good, and worthless test plotted on the same graph. The accuracy of the test depends on how well the test separates the group being tested into those with and without the matching question (whether two citations belong to the same author or not). In Figure 5.1 ROC curve reflects the appropriate accuracy of RF classifier in predicting negative and positive author's citation pairs.

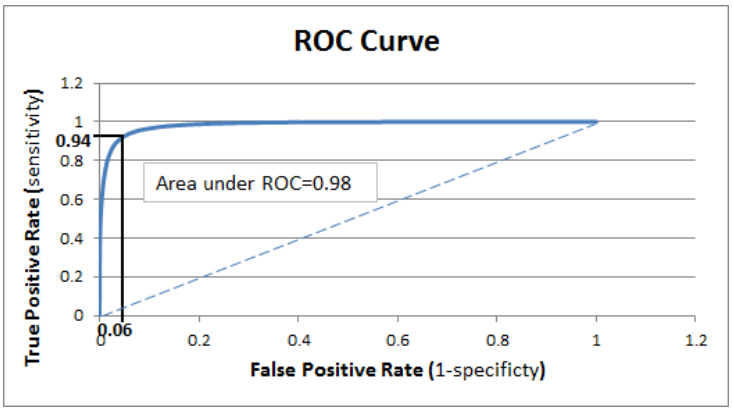

Figure 5.1: ROC curve for predicting the author's citation pairs.

5.3 Random forest versus other machine learning approaches

Out of four algorithms RF shows the highest accuracy i.e. 94.78% than the other three. It depicts that RF performs better and gives good prediction results than the others. From the Table 5.4 the algorithms namely J48, DNN and Nive Bayes have 92.32%, 87.63% and 85.01% accuracy respectively. The 94.78% accuracy means our algorithm will give high results. The remaining 5.22% incorrect results will be displayed. Despite this incorrect results RF is much better than the other algorithms which show 7.68%, 12.37% and 14.99% incorrect results respectively.

Table 5.4: Analysis of algorithm on Knowledge level ORCID Data set

Classification Techniques	Total Number of Instances	Correctly Classified Instance	Incorrectly Classified instance	Kappa Statistic	Mean absolute error	Root mean squared error	Accuracy
RF	200,777	190,291	10,486	0.87	0.09	0.20	94.78%
J48	200,777	185,616	15,161	0.80	0.12	0.26	92.32%
DNN	200,777	175,949	24,828	0.69	0.18	0.31	87.63%
Naive Bayes Classifier	200,777	170,735	30,042	0.63	0.15	0.37	85.01%

The results that have been obtained from these comparative studies show that RF has the highest accuracy. Kappa is near to perfect which renders it as the most capable classifier that can be recommended. Another characteristic of RF is that it has the lowest squared error relative to the other three algorithms and has the least number of incorrect results.

Table 5.5 shows the detailed accuracy results that have been obtained from the comparison between RF, J48, DNN and Nive Bayes. All measurements proved that RF is performing better than others. True Positive Rate (TPR) or sensitivity in RF is equal to 0.97 for class "No" and the False Positive Rate (FPR) or the specificity is equal to 0.03 for class "Yes". That means 0.03 of negative citation records are incorrectly classified as positive pairs. Accordingly, 0.12 of positive citation pairs are classified as negative citation pairs while they are positive pairs. Precision

shows that RF succeeded to classify 0.96 of negative citation pairs that belong to different authors and 0.93 as positive citation pairs that belongs to the same authors.

ROC curves show that RF performs better than others for both negative and positive citation records. According to PRC area RF succeeded to predict the citations that belong to the same author while PRC area value in J48 , DNN and Nive Bayes are 0.87, 0.83 and 0.81 respectively show that these approaches are overfitting to predict positive citation records.

Table 5.5: Detailed Accuracy by Class

Classification Techniques	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	PRC Area	Class
RF	0.97	0.12	0.96	0.97	0.96	0.98	0.99	No
	0.88	0.03	0.93	0.88	0.90	0.98	0.97	Yes
J48	0.96	0.17	0.94	0.96	0.95	0.94	0.96	No
	0.84	0.04	0.87	0.84	0.86	0.94	0.87	Yes
DNN	0.92	0.25	0.91	0.92	0.92	0.90	0.94	No
	0.75	0.08	0.80	0.75	0.77	0.90	0.83	Yes
Naive Bayes Classifier	0.90	0.27	0.90	0.90	0.90	0.90	0.95	No
	0.73	0.10	0.73	0.73	0.73	0.90	0.81	Yes

The ROC curve in Figure 5.2 shows that RF achieves the highest area under curve value equal to 0.98 comparing with J48 , DNN and Nive Bayes which were recorded 0.94, 0.90 and 0.90 respectively.

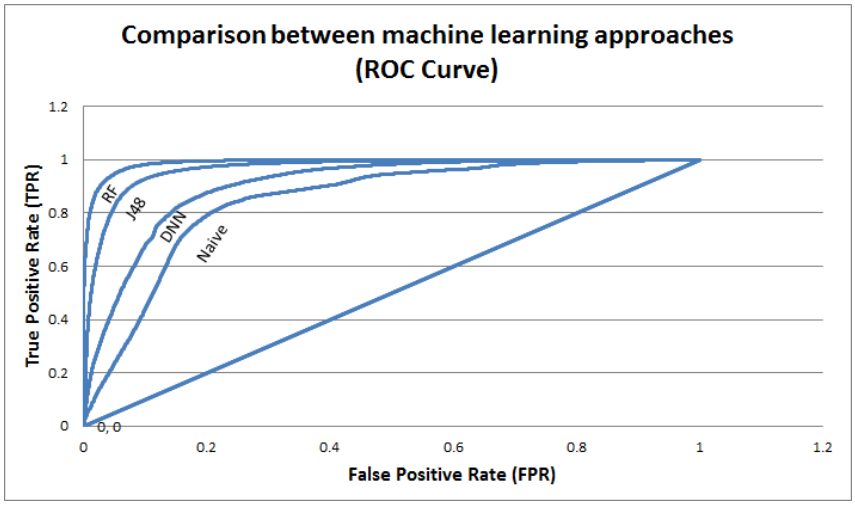

Figure 5.2: Comparison between machine learning approaches (ROC Curve)

5.4 Features Ranking and Distance Matrices

The results in Table 5.6 show the ranking of Distance Matrices (DM). The InfoGain values of DM that we used to find the similarity of coauthors pairs have the highest values between 0.278 and 0.238 excluding NGram algorithm. The InfoGain gives us an indication that coauthors are the most important feature among others in the citation records. Using semantic technique to find the similarity between work title pairs ranked them as the second important feature after

coauthors. Journal title is the third and publishing year is the least important with InfoGain value equal to 0.033. Ranking table also shows that using Jaro Winkler and Levenshtein algorithms to find the similarity of coauthors pairs is worth more than using them with journal title but using Cosine and Jaccard algorithms to find the similarity of journal title pairs is more accurate than using them with coauthors.

Table 5.6: Features Ranking and Distance Matrices

Attributes	DM	Ranked DM	Rank	InfoGain
Coauthoers	Jaro Winkler	2	1	0.278
	Levenshtein	4	2	0.256
	Cosine	3	3	0.252
	Jaccard	1	4	0.238
	NGram	5	14	0.024
Work Title	Title2 using collection of dictionaries (En-Wikipedia, En-PubMedOA, En-BNC)	14	5	0.118
	Title1 using WordNet dictionary	13	6	0.104
Journal Title	Sorensen-Dice	11	7	0.060
	Cosine	8	8	0.059
	Jaccard	6	9	0.059
	Normalized Levenstein	12	10	0.045
	Jaro Winkler	7	11	0.039
	NGram	10	12	0.035
	Levenshtein	9	13	0.033
Year	Year	15	15	0.002

In order to evaluate the importance of each feature, we implemented different experiments using RF with respect to the parameters $F_{num} = 4$, $T_{num} = 300$ and $T_{depth} = 0$. Figure 5.3 shows the accuracy that RF achieved by excluding one feature periodically. Removing coauthors will decrease the accuracy to 82.10% which has an impact of 12.9% in eliminating ambiguity in author's names while removing journal title, work title and publishing year will decrease the accuracy to 89%, 91.44% and 91.80% respectively.

These experiments showed that, coauthors are the most important feature among others in citation records in solving author names disambiguation problem. Journal title is the second feature that has an important impact then work title and finally the publishing year. All of these features have a strong and a promising estimation when they work together in solving the ambiguity of author names.

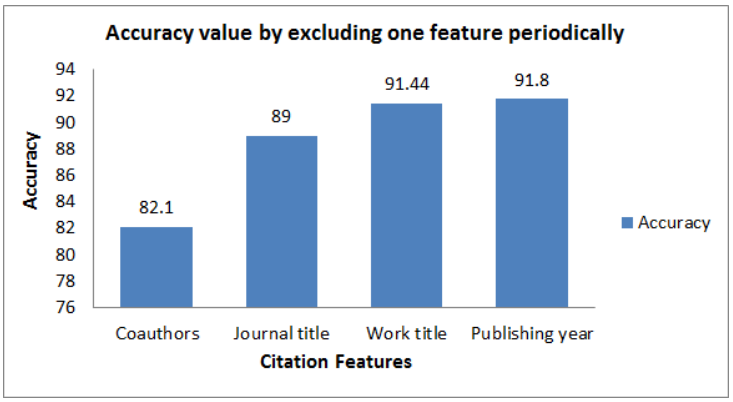

Figure 5.3: Accuracy value by excluding one feature periodically

We excluded the least two values (Year and Levenshtein which used with journal title) according to InfoGain results in Table 5.6 then we ranked the features again using InfoGain measurement. The results in Table 5.7 show that coauthors are the most important feature among others, work title is the second and journal title is the third.

Table 5.7 also shows that using Levenshtein algorithm to find the similarity of coauthors pairs is worth more than using it with journal title but using Cosine and Jaccard algorithms to find the similarity of journal title and coauthors pairs has the same importance.

Table 5.7: Features Ranking and Distance Matrices by excluding the two smallest InfoGain values

Attributes	DM	Ranked DM	Rank	InfoGain
Coauthoers	cLeven	4	1	0.10074
Work Title	Title2 using collection of dictionaries (En-Wikipedia, En-PubMedOA, En-BNC)	13	2	0.09042
Coauthoers	cJaccard	1	3	0.08972
	cCosine	3	4	0.08615
	cJaro	2	5	0.05791
Journal Title	jJaccard	5	6	0.03345
	jCosine	7	7	0.03161
Work Title	Title1 using WordNet dictionary	12	8	0.02601
Journal Title	jSorensenDice	10	9	0.02524
	jNormlizedLev	11	10	0.01083
	jJaro	6	11	0.00991
	jNGram	9	12	0.00817
	jLeven	8	13	0.00682

We evaluate the importance of the features based on the InfoGain ranking results in Table 5.7. For this purpose we implemented different experiments using RF with respect to the parameters $F_{num} = 4$, $T_{num} = 300$ and $T_{depth} = 0$. We calculated the accuracy of RF classifier model by validating the predicted results against the testing set using all the distance matrices in Table 5.7 that has been ranked by InfoGain measurement for each attribute. Table 5.8 and 5.9 show

that we can distinguish whether two citations belong to the same author or not even without the publishing of the year with high accuracy.

Table 5.8: The statistics results in the testing phase

#	Test	Result
1	Correctly Classified Instances	92.64%
2	Incorrectly Classified Instances	7.36%
3	Kappa statistic	0.81
4	Mean absolute error	0.11
5	Root mean squared error	0.24
6	Total Number of Instances	200,776

Table 5.9: RF detailed accuracy results in testing phase

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	PRC Area
no	0.96	0.15	0.94	0.96	0.95	0.97	0.98
yes	0.85	0.04	0.88	0.85	0.87	0.97	0.93

Figure 5.4 shows the accuracy that RF achieved by excluding one feature periodically. These experiments showed that coauthors are the most important feature among others in citation records in solving author's names disambiguation problem. Work title is the second important feature and finally journal title.

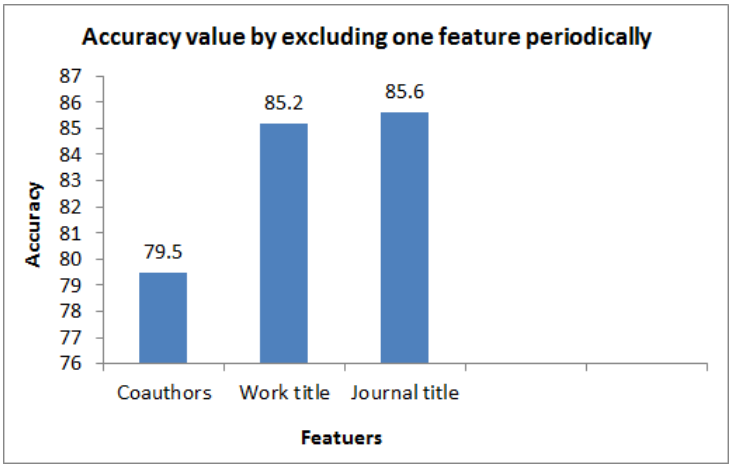

Figure 5.4: Accuracy value by excluding one feature periodically according to InfoGain ranking results

5.5 Execution Time

Table 5.10 represents various data-sets that have been chosen randomly from ORCID database. We extract all citation combinations from each dataset that share LNFI name strings. The execution time that we needed before constructing the machine learning model is divided into

two parts. The former is the time that we need for extracting the features from ORCID database and the latter is the time we need to find the similarity between negative and positive citation pairs.

Table 5.10: Pre-processing time performance

LNFI	Positive citations	Execution time per second	Negative citations	Execution time per second	Similarity calculation time per minute	Total execution time per minute	Total citations
wang, y	4584	225.83	1090	192.83	0.13	7.11	5674
Ali, M	4052	210.30	12804	235.59	0.42	7.86	16856
Park, J	5886	210.32	43572	336.02	1.06	10.17	49458
Chen, Y	42660	311.49	208328	832.87	5.19	24.26	250988
Liu, Y	35650	305.87	267500	1001.61	5.87	27.66	303150
Zhang, Y	120166	626.08	546518	2025.93	12.71	56.91	666684
Li, J	456362	1675.95	355850	1670.76	16.29	72.06	812212
Kim, J	330364	1159.75	850450	2603.62	22.15	84.88	1180814
Lee, J	764264	2394.13	646306	3210.76	28.09	121.50	1410570

The quadratic curve in Figure 5.5 represents the relationship between the total number of extracted citations and the total time that consists of the execution time and the similarity time.

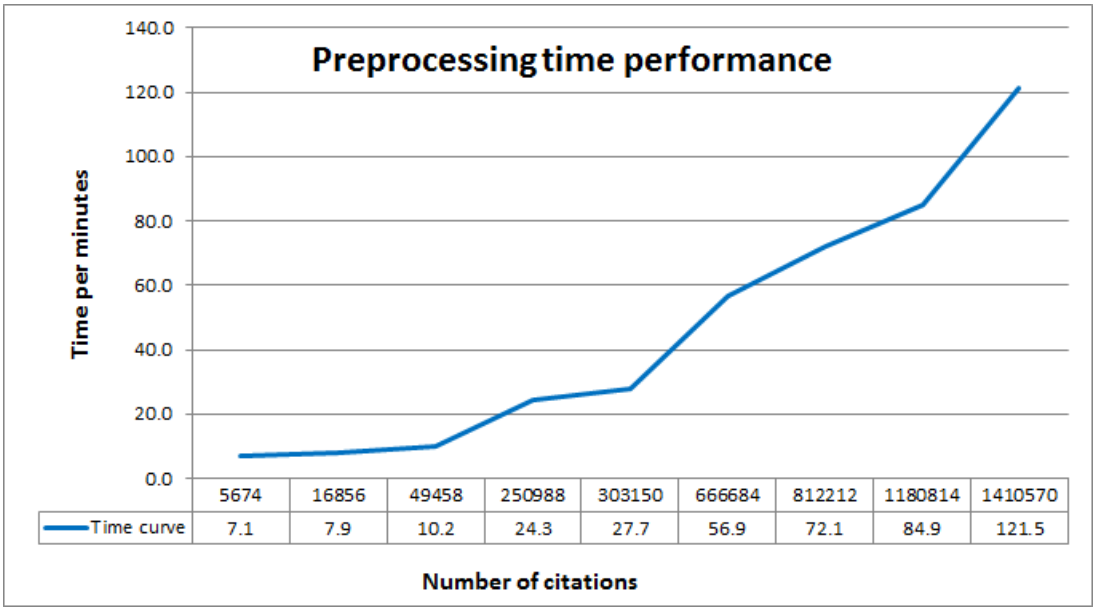

Figure 5.5: Time performance in pre-processing phase

It is also necessary to take into account the time required by the RF learned model to predict unseen samples. For this purpose different testing data sets have been used for evaluating the classification model. For each new test set we calculated the accuracy and the time that RF spent to predict these new citations records. Figure 5.6 illustrates the time performance of RF classifier for each data set. It describes the execution time required to predict new citation records. We found a non-linear direct proportion between the citation pairs and the execution time and that means more accurate prediction requires more trees, which results in a slower model. We conclude that RF algorithm is fast enough in solving the AD problem based on ORCID citations.

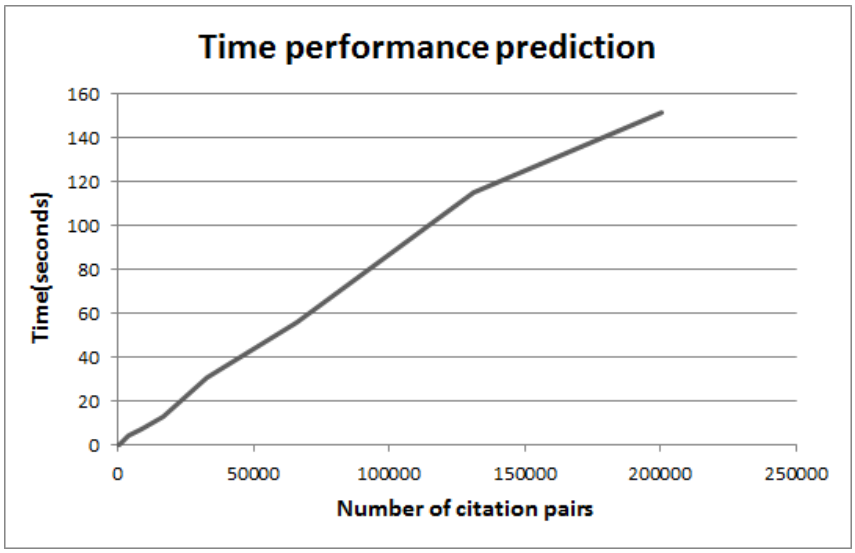

Figure 5.6: Time performance prediction using RF classifier

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have addressed the problem of author's names disambiguation for different authors with the same name on ORCID database citations. In order to solve the problem, we introduced a set of methodological steps to exploit the relationships between citations. We have discovered the implied topic-based relationships in citations to influence author names ambiguity and that the accuracy of disambiguation can be significantly improved by looking at citations in ORCID databases or other publications in different databases.

More than one million citation pairs were extracted from ORCID database to construct the negative and positive citation pairs to be used in our experiments. Positive pairs mean that these citations belong to the same author while negative pairs belong to different authors'.

The methodology of solving the problem of author's names was based on citations that consists of coauthors names, journal title, work title and publishing year. Different distance matrices were used to find the similarity between coauthors and journal title. Semantic knowledge based on collective English language dictionaries was used to find how much two work titles are similar or not. Author who publishes two publications in the same year gives a strong indication that these publications belong to the same author.

We believe that machine learning approaches are the key for solving this problem. Different machine learning algorithms were used to classify the citations record. Our experiment results show that the disambiguation accuracy improved to 94.78% by using RF classifier while Nive Bayes, J48 and DNN have 85.01%,92.32% and 87.63% accuracy respectively.

Of the four basic attributes, the coauthor attribute provides the most useful information for disambiguation, and journal title is slightly better than work title and the year of publishing. In addition, disambiguation information derived by Jaro Winkler, Levenshtein, Cosine and Jaccard metric contains less noise in finding the similarity of coauthors than NGram. Sorensen-Dice, Cosine, Jaccard and Normalized Levenstein metric have less noise in finding the similarity of journal titles.

Also, we conclude that removing the year of publishing will not negatively affect the result

of disambiguation accuracy. Almost, Rf achieved 93% accuracy and the ROC curve reflects the appropriate accuracy of 97% in predicting negative and positive author's citation pairs.

The Web application, which was designed based on our vision to solve the problem of ambiguity in authors' names, is a sure indication of the usefulness of the methodology of our study that has the ability to classify new citations outside the boundaries of ORCID databases.

In summary, our contribution in grouping citation pairs of the same author into the correct class is more accurate and proposes a useful solution for name disambiguation improvements.

6.2 Future Work

Different experiments have been left for the future due to lack of time i.e. the experiment requiring even months to finish. Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods. There are some ideas that I would have liked to try during the description and the development of the functionality of solving author's names disambiguation problem.

This thesis has been mainly focused on the use of ORCID citations for author's names matching, and most of the mechanisms used are focused on answering the question whether two citations belong to the same author or not. The following points among the points that we have been achieved are left outside the scope of this thesis

- Filtration ORCID identifiers in which each author must have a unique and distinct ORCID ID. This assumption will help ORCID to:
  - Achieve the one-to-one relationship between ORCDI ID and author name.
  - Save more space by removing the duplication records.
  - Solve the problem of name disambiguation and achieve more accuracy.
- Updating the work activity records for each author will help ORCID to:
  - Add a new external citation to the appropriate author records.
  - Control over profile information from third-party profiles.
- Trying other existent larger databases, such as DBLP, ACM Digital Library and Citeseer.
- Author citation information, the key input to our system, is still imperfect due to some information loss. Considering the variety of cultural backgrounds of ORCID citation authors, a better system is necessary to recover information lost during the data collection stages such as email and affiliation.

Appendix A

Web-Application

Figure A.1 illustrates the *matching* and *not matching* process for any citation pairs. It starts by checking whether two authors belong to the LNFI criteria or not. If they are different then the results will not match. If they belong to the LNFI criteria then the process will continue to calculate the similarity of the coauthors, journal title, work title and the year of publications. The application will pass the similarity vector to the trained RF learning model which gives us an indication whether the two citations records are related to the same author or not. Different java classes have been used in the application as follow:

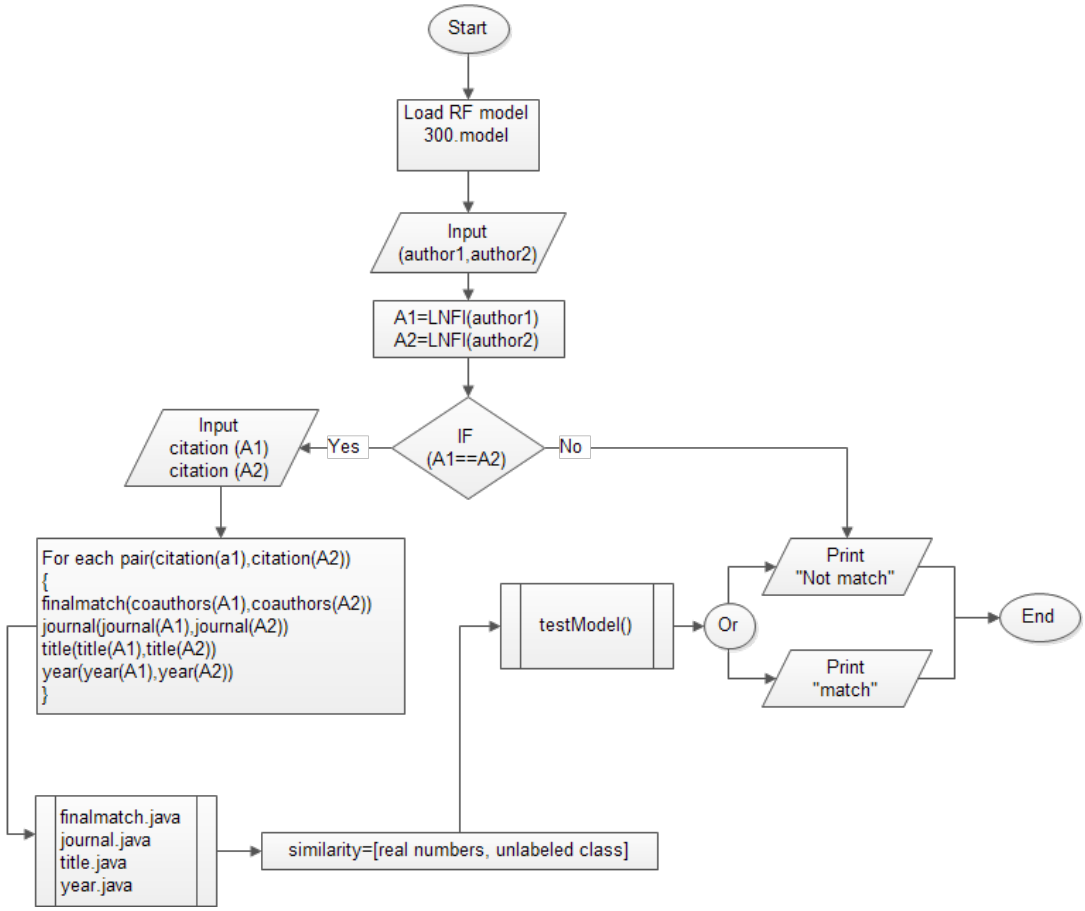

Figure A.1: Matching and not matching web-application process

The *processRequest* method is used to declare the desired variables and the required methods that we use to perform the calculation process.

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      DecimalFormat formatter = new DecimalFormat("#.##");
      double j[] = {0, 0, 0, 0, 0, 0, 0};
      double c[] = {0, 0, 0, 0, 0};
      String finalscore[] = {""};
      String prediction = "";
      PrintWriter out = response.getWriter();
      String auhtor1firstname =
         request.getParameter("authoronefirstname").toString();
      String author1lastname =
         request.getParameter("authoronelastname").toString();
      String author2firstname =
         request.getParameter("authortwofirstname").toString();
      String author2lastname =
         request.getParameter("authortwolastname").toString();
      String fullname = "[" + auhtor1firstname + " " +
         author1lastname + "]<==>[" + author2firstname + " " +
         author2lastname + "] is :";
      int y1 =
         Integer.parseInt(request.getParameter("firstyear").toString());
      int y2 =
         Integer.parseInt(request.getParameter("secondyear").toString());
      String c1 = request.getParameter("co1").toString();
      String c2 = request.getParameter("co2").toString();
      String title1 =
         request.getParameter("authoronetitle").toString();
      String title2 =
         request.getParameter("authortwotitle").toString();
      String journal1 =
         request.getParameter("authoronejournal").toString();
      String journal2 =
         request.getParameter("authortwojournal").toString();
      double yearvalue = 0;
      double worktitle = 0;
      double worktitle2 = 0;
      if (author1lastname.equals(author2lastname) &&
```

```
    auhtor1firstname.substring(0,
    1).equals(author2firstname.substring(0, 1))) {
    yearvalue = years(y1, y2);
    worktitle = maintitle(title1, title2);
    worktitle2 = smain(title1, title2);
    j = journal.journalmatch(journal1, journal2);
    c = finalmatch.comatch(c1, c2);
    finalscore[0] = formatter.format(c[0]) + "," +
       formatter.format(c[1]) + "," + formatter.format(c[2]) +
       ","
          + formatter.format(c[3]) + "," +
             formatter.format(c[4]) + ","
          + formatter.format(j[0]) + "," +
             formatter.format(j[1]) + "," +
             formatter.format(j[2]) + ","
          + formatter.format(j[3]) + "," +
             formatter.format(j[4]) + "," +
             formatter.format(j[5]) + ","
          + formatter.format(j[6]) + ","
          + formatter.format(worktitle) + "," +
             formatter.format(worktitle2) + ","
          + formatter.format(yearvalue) + "," + "?";
} else {
    finalscore[0] = 0 + "," + 0 + "," + 0 + ","
          + 0 + "," + 0 + ","
          + 0 + "," + 0 + "," + 0 + ","
          + 0 + "," + 0 + "," + 0 + ","
          + 0 + ","
          + 0 + "," + 0 + ","
          + 0 + "," + "?";
}
writeCsvFile(finalscore[0]);
unlabel(fullname);
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet GreetingServlet</title>");
out.println("</head>");
out.println("<body>");
out.println("<p>Welcome " + formatter.format(c[0]) + "," +
    formatter.format(c[1]) + "," + formatter.format(c[2]) + ","
       + formatter.format(c[3]) + "," + formatter.format(c[4])
          + ","
```

```
            + formatter.format(j[0]) + "," + formatter.format(j[1])
               + "," + formatter.format(j[2]) + ","
            + formatter.format(j[3]) + "," + formatter.format(j[4])
               + "," + formatter.format(j[5]) + ","
            + formatter.format(j[6]) + ","
            + formatter.format(worktitle) + "," +
               formatter.format(worktitle2) + ","
            + formatter.format(yearvalue) + "," + "?" + "</p><br>");
      out.println(readfile() + "</body>");
      out.println("</html>");

      out.close();
   }
```

The following *finalmatch* class is used to calculate the coauthors similarity scores

```
public class finalmatch {
   public static Jaccard j;
   public static JaroWinkler jw;
   public static OptimalStringAlignment OP;
   public static Levenshtein we;
   public static SorensenDice so;
   public static NGram ng;
   public static Cosine co;
   public static Damerau dam;
   public static NormalizedLevenshtein Norl;
   public static String normlizetext(String w) {
      w = w.replaceAll("\\s+$", "");//rightRemoved
      w = w.replaceAll("^\\s+", "");//leftRemoved
      w = w.replace(",", " ");
      w = w.toLowerCase();
      // System.out.print(w);
      return w;
   }
   public static double jaccardcomp(String w1, String w2) {
      double r = 0;
      r = (double) (j.similarity(w1, w2));
      return r;
   }
   public static double jarocomp(String w1, String w2) {
      double r = 0;
      r = (double) (jw.similarity(w1, w2));
```

```
        return r;
    }
    public static double Cosinecomp(String w1, String w2) {
        double r = 0;
        r = (double) (co.similarity(w1, w2));
        return r;
    }
    public static double Levencomp(String w1, String w2) {
        double r = 0;
        r = (double) (Norl.similarity(w1, w2));
        return r;
    }
    public static double Dameraucomp(String w1, String w2) {
        double r = 0;
        r = (double) (ng.distance(w1, w2));
        return r;
    }
    public static double printgridarray(double a[][], int xlen, int
        ylen, String str) {
        int fullmatchcount = 0;
        int partialcount = 0;
        double partialsum = 0;
        double result = 0;
        double partialweight = 0;
        double fullmatchweight = 0;
        double f1 = 0, f2 = 0;
        int print = 0;// 1 to desplay the details and 0 the final
            results
        double pweight = .75;
        if (print == 1) {
            System.out.printf("\n=========\n");
            System.out.printf("\t\t%s\t\t", str);
            System.out.printf("\n----------\n");
        }
        for (int x = 0; x <= xlen; x++) {
            for (int y = 0; y <= ylen; y++) {

                if (print == 1) {
                    System.out.printf("%.2f ", a[x][y]);
                }
                if (a[x][y] == 1) {
                    fullmatchcount++;
```

```
            }
            if (a[x][y] != 1 && a[x][y] > pweight) {
               partialsum = partialsum + (a[x][y] - pweight);
               partialcount++;
            }
         }
         if (print == 1) {
            System.out.printf("\n");
         }
      }
      if (fullmatchcount == 0 && partialcount == 0) {
         f1 = 0;
         f2 = 0;
         fullmatchweight = f1;
         partialweight = f2;
      } else if (fullmatchcount == 0 && partialcount > 0) {
         f1 = 0;
         f2 = 0.5;
         fullmatchweight = f1;
         partialweight = ((partialsum / partialcount));

      } else if (fullmatchcount == 1 && partialcount > 0) {
         f1 = 0.5;
         f2 = 0.5;
         fullmatchweight = f1;
         partialweight = ((partialsum / partialcount));
      } else if (fullmatchcount == 1 && partialcount == 0) {
         f1 = 0.5;
         f2 = 0;
         fullmatchweight = f1;
         partialweight = f2;

      } else if (fullmatchcount == 2 && partialcount > 0) {
         f1 = 0.75;
         f2 = 0.25;
         fullmatchweight = f1;
         partialweight = ((partialsum / partialcount));

      } else if (fullmatchcount == 2 && partialcount == 0) {
         f1 = 0.75;
         f2 = 0;
         fullmatchweight = f1;
```

```
            partialweight = f2;
        } else if (fullmatchcount > 2) {
            f1 = 1;
            f2 = 0;
            fullmatchweight = f1;
            partialweight = f2;
        }
        result = fullmatchweight + partialweight;
        if (print == 1) {
            System.out.printf("\n---------------\n");
            System.out.printf("f1:(%.2f)\tFullmatch count:(%d)
                ,\t\tFullmatch weight:%.2f", f1, fullmatchcount,
                fullmatchweight);
            System.out.printf("\nf2:(%.2f)\tPartial sum(%.2f),partial
                count:(%d), Partial Weight:%.2f", f2, partialsum,
                partialcount, partialweight);
            System.out.printf("\n\t\t\t\t\t\t Result:\t%.2f\t", result);
            System.out.printf("\n=================\n");
        }
        return result;
    }
    public static double[] comatch(String s1, String s2) {
        int i;
        jw = new JaroWinkler();
        j = new Jaccard(1);
        OP = new OptimalStringAlignment();
        co = new Cosine();
        ng = new NGram();
        we = new Levenshtein();
        so = new SorensenDice();
        Norl = new NormalizedLevenshtein();
        dam = new Damerau();
        double co[] = {0, 0, 0, 0, 0};
        String beforew1 = "";
        String afterw1 = "";
        String beforew2 = "";
        String afterw2 = "";
        String secondS = "";
        String firstS = "";
        double c1 = 0.0;
        double c2 = 0.0;
        double c3 = 0;
```

```
double c4 = 0;
double c5 = 0;
String[] s1arry = s1.split(";");
String[] s2arry = s2.split(";");
double outgrid[][] = new double[s1arry.length][s2arry.length];
double outgrid1[][] = new double[s1arry.length][s2arry.length];
double outgrid2[][] = new double[s1arry.length][s2arry.length];
double outgrid3[][] = new double[s1arry.length][s2arry.length];
double outgrid4[][] = new double[s1arry.length][s2arry.length];
int x = 0;
int y = 0;
if (s1.isEmpty() || s2.isEmpty()) {
} else {
   for (String w1 : s1arry) {
      y = 0;
      w1 = normlizetext(w1);
      if (w1.contains(",")) {
         beforew1 = w1.split(",")[0];
         afterw1 = w1.split(",")[w1.split(",").length - 1];
         firstS = beforew1.concat(" ").concat(afterw1);
      } else {
         firstS = w1;
      }
      for (String w2 : s2arry) {
         w2 = normlizetext(w2);
         if (w2.contains(",")) {
            beforew2 = w2.split(",")[0];
            afterw2 = w2.split(",")[w2.split(",").length - 1];
            secondS = beforew2.concat(" ").concat(afterw2);
         } else {
            secondS = w2;
         }
         c1 = jaccardcomp(w1, w2);
         c2 = jarocomp(w1, w2);
         c3 = Cosinecomp(w1, w2);
         c4 = Levencomp(w1, w2);
         c5 = Dameraucomp(w1, w2);
         outgrid[x][y] = c1;
         outgrid1[x][y] = c2;
         outgrid2[x][y] = c3;
         outgrid3[x][y] = c4;
         outgrid4[x][y] = c5;
```

```
                y++;
            }
            x++;
        }
        co[0] = printgridarray(outgrid, x - 1, y - 1, "Jacard");
        co[1] = printgridarray(outgrid1, x - 1, y - 1, "Jaro");
        co[2] = printgridarray(outgrid2, x - 1, y - 1, "Cosine");
        co[3] = printgridarray(outgrid3, x - 1, y - 1, "Leven");
        co[4] = printgridarray(outgrid4, x - 1, y - 1, "Damerau");
    }
    return co;
  }
}
```

The following *journal* class is used to calculate the journal titles similarity scores

```
public class journal {
   public static Jaccard j;
   public static JaroWinkler jw;
   public static OptimalStringAlignment OP;
   public static Levenshtein we;
   public static SorensenDice so;
   public static NGram ng;
   public static Cosine co;
   public static Damerau dam;
   public static NormalizedLevenshtein Norl;
   public static WeightedLevenshtein wlv;
   public static double jaccardcomp(String w1, String w2) {
      double r = 0;
      r = (double) (j.similarity(w1, w2));
      return r;
   }
   public static double jarocomp(String w1, String w2) {
      double r = 0;
      r = (double) (jw.similarity(w1, w2));
      return r;
   }
   public static double Cosinecomp(String w1, String w2) {
      double r = 0;
      r = (double) (co.similarity(w1, w2));
      return r;
   }
```

```
public static double Levencomp(String w1, String w2) {
    double r = 0;
    r = (double) (we.distance(w1, w2));
    return r;
}
public static double Dameraucomp(String w1, String w2) {
    double r = 0;
    r = (double) (dam.distance(w1, w2));
    return r;
}
public static double OPcomp(String w1, String w2) {
    double r = 0;
    r = (double) (OP.distance(w1, w2));
    return r;
}
public static double Ngramcomp(String w1, String w2) {
    double r = 0;
    r = (double) (ng.distance(w1, w2));
    return r;
}
public static double SorensenDicecomp(String w1, String w2) {
    double r = 0;
    r = (double) (so.distance(w1, w2));
    return r;
}
public static double Normalcomp(String w1, String w2) {
    double r = 0;
    r = (double) (Norl.distance(w1, w2));
    return r;
}
public static double wlvcomp(String w1, String w2) {
    double r = 0;
    r = (double) (wlv.distance(w1, w2));
    return r;
}
public static double[] journalmatch(String string1, String
    string2) {
    j = new Jaccard(4);
    jw = new JaroWinkler(4);
    co = new Cosine(4);
    we = new Levenshtein();
    dam = new Damerau();
```

```
        OP = new OptimalStringAlignment();
        ng = new NGram(4);
        so = new SorensenDice();
        Norl = new NormalizedLevenshtein();
        double j[] = {0, 0, 0, 0, 0, 0, 0};
        int i = 0;
        double s1, s2, s3, s4, s5, s6, s7, s8, s9 = 0;
        DecimalFormat decimalFormat = new DecimalFormat("#.00");
        s1 = jaccardcomp(string1, string2);//jaccrd
        s2 = jarocomp(string1, string2);//jaro
        s3 = Cosinecomp(string1, string2);//Cosinecomp
        s4 = Levencomp(string1, string2);//Levencomp
        s5 = Dameraucomp(string1, string2);//Dameraucomp
        s6 = OPcomp(string1, string2);//OptimalStringAlignment
        s7 = Ngramcomp(string1, string2);//NGram
        s8 = SorensenDicecomp(string1, string2);//SorensenDice
        s9 = Normalcomp(string1, string2);//NormalizedLevenshtein
        j[0] = s1;
        j[1] = s2;
        j[2] = s3;
        j[3] = s4;
        j[4] = s7;
        j[5] = s8;
        j[6] = s9;
        return j;
    }
}
```

The following *title* class is used to calculate the work titles similarity score using collection of dictionaries.

```
public class title {
    String a = "";
    String b = "";
    public Compare(String a, String b) {
        this.a = a.toLowerCase();
        this.b = b.toLowerCase();
    }
    public double getResult() {
        StringTokenizer ta = new StringTokenizer(a,
            "~!@#$%^&*()_-=+`[]{}:\"\\|;'<>?,./ ");
        StringTokenizer tb = new StringTokenizer(b,
```

```
        "~!@#$%^&*()_-=+`[]{}:\"\\|;'<>?,./ ");
    if (ta.countTokens() == 1) {
        File f = new
            File("C:\\MyFirstServlet1\\semantics\\src\\collocation_dict\\"
            + Character.toString(a.charAt(0)) + "\\" +
            Character.toString(a.charAt(1)) + "\\" + a + ".txt");
        if (!f.exists()) {
            return -1;
        }
    }
    if (tb.countTokens() == 1) {
        File f = new
            File("C:\\MyFirstServlet1\\semantics\\src\\collocation_dict\\"
            + Character.toString(b.charAt(0)) + "\\" +
            Character.toString(b.charAt(1)) + "\\" + b + ".txt");
        if (!f.exists()) {
            return -1;
        }
    }
    String aa = "";
    String bb = "";
    while (ta.hasMoreTokens()) {
        String g = ta.nextToken();
        String add = read(g);
        if (add.equals("")) {
            aa = aa + " " + g;
            //Jumah
            //System.out.print(aa);

        } else {
            aa = aa + " " + read(g);
        }
    }
    while (tb.hasMoreTokens()) {
        String g = tb.nextToken();
        String add = read(g);
        if (add.equals("")) {
            bb = bb + " " + g;
            //Jumah
            //  System.out.print(bb);
        } else {
            bb = bb + " " + read(g);
```

```
            }
        }
        try {
            TermVectorStorage storage = new HashMapTermVectorStorage();
            VectorClassifier vc = new VectorClassifier(storage);
            vc.teachMatch("category", aa);
            double result = vc.classify("category", bb);
            return result;
        } catch (Exception e) {
            return -1;
        }
    }
    private String read(String g) {
        try {
            String filename =
                ("C:\\MyFirstServlet1\\semantics\\src\\collocation_dict\\"
                + Character.toString(g.charAt(0)) + "\\" +
                Character.toString(g.charAt(1)) + "\\" + g + ".txt");
            File f = new File(filename);
            if (f.exists()) {
                BufferedReader bread = new BufferedReader(new
                    FileReader(filename));
                String rt = bread.readLine();
                bread.close();
                return rt;
            } else {
                return "";
            }
        } catch (Exception e) {
            return "";
        }
    }
}
```

The following *years* method is used to calculate the year of publications similarity score.

```
public static double years(int year1, int year2) {
        int result = 0;
        double finalresult = 0;
        result = Math.abs(year2 - year1);
        if (result == 0) {
            finalresult = 1;
```

```
        } else {
            finalresult = (double) 1 / result;
        }
        return finalresult;
    }
```

The following *loadModel* method is used to load RF trained method.

```
public static RandomForest loadModel(String fileName) {
        try {
            int bufferSize = 16 * 1024;
            ObjectInputStream in = new ObjectInputStream(new
                BufferedInputStream(new FileInputStream(fileName),
                bufferSize));
            Object tmp = in.readObject();
            RandomForest classifier = (RandomForest) tmp;
            in.close();
            loadfile = "===== Loaded model: " + fileName + " =====";
            return classifier;
        }
        catch (Exception e) {
            loadfile = "Problem found when reading: " + fileName;
        }
        return null;
    }
```

The following *unlabel* method is used to predict the unseen citation pairs.

```
public void unlabel(String name) {
        unlabeledtest = "C:\\MyFirstServlet1\\test\\test.arff";
        String unclasses = "";
        String MoM = "";
        String y = "yes";
        String n = "no";
        String score = "";
        try {
            Instances unlabeled = new Instances(new BufferedReader(new
                FileReader(unlabeledtest)));
            unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
            Instances labeled = new Instances(unlabeled);
            for (int i = 0; i < unlabeled.numInstances(); i++) {
                double clsLabel =
                    randomForest.classifyInstance(unlabeled.instance(i));
```

```
            double[] prediction =
               randomForest.distributionForInstance(unlabeled.instance(i));
            labeled.instance(i).setClassValue(clsLabel);
            String ClassLabel;
            ClassLabel = unlabeled.classAttribute().value((int)
               clsLabel);
            unclasses = "\n" + unclasses + (i + 1) + "\t" +
               ClassLabel;
            if (ClassLabel.equals(y)) {
               MoM = "Mach";
            } else if (ClassLabel.equals(n)) {
               MoM = "Not Match";
            }
            score = Arrays.toString(prediction);
         }
         writeoutput(name, MoM, score);
         BufferedWriter writer = new BufferedWriter(
               new
                  FileWriter("C:\\MyFirstServlet1\\unlabel\\labeled.arff"));
         writer.write(labeled.toString());
         writer.newLine();
         writer.flush();
         writer.close();
      } catch (Exception e) {
         System.out.println("Problem found when writting: " +
            unlabeledtest);
      }
   }
```

Figure A.2 is a graphical user interface web-application used to check whether two authors' share the same family name and the first initial of the given name. The application window is split into two sub-windows one on the right and one on the left. Each one represents an author meta-data that consist of author LNFI, coauthors, journal title and work title. All records are required during data entry. The *Submit* button at the top left corner is used to printout the results of the application and the *About* button is used to printout the instructions about how to use the application. The results are displayed at the bottom of the browser screen.

Figure A.2: ML Based Disambiguation of Author's Names in ORCID Citations Application

Bibliography

[1] Orcid organization. Researcher profile.

[2] Dongwon Lee, Jaewoo Kang, Prasenjit Mitra, C Lee Giles, and Byung-Won On. Are your citations clean. *Communications of the ACM*, 50(12):33–38, 2007.

[3] Kai-Hsiang Yang, Hsin-Tsung Peng, Jian-Yi Jiang, Hahn-Ming Lee, and Jan-Ming Ho. Author name disambiguation for citations using topic and web correlation. *Research and advanced technology for digital libraries*, pages 185–196, 2008.

[4] Neil R Smalheiser and Vetle I Torvik. Author name disambiguation. *Annual Review of Information Science and Technology*, pages 5–34, 2009.

[5] George Papadakis and Georgios Paliouras. Mycites: An intelligent information system for maintaining citations. *Artificial Intelligence: Theories, Models and Applications*, pages 371–376, 2008.

[6] Prasenjit Mitra, Jaewoo Kang, Dongwon Lee, and Byung-won On. Comparative study of name disambiguation problem using a scalable blocking-based framework. In *Digital Libraries, 2005. JCDL'05. Proceedings of the 5th ACM/IEEE-CS Joint Conference on*, pages 344–353. IEEE, 2005.

[7] Jan Youtie, Stephen Carley, Alan L Porter, and Philip Shapira. Tracking researchers and their outputs: new insights from orcids. *Scientometrics*, 113(1):437–453, 2017.

[8] Laurel L Haak, Martin Fenner, Laura Paglione, Ed Pentz, and Howard Ratner. Orcid: a system to uniquely identify researchers. *Learned Publishing*, 25(4):259–264, 2012.

[9] Hui Han, Lee Giles, Hongyuan Zha, Cheng Li, and Kostas Tsioutsiouliklis. Two supervised learning approaches for name disambiguation in author citations. In *Digital Libraries, 2004. Proceedings of the 2004 joint ACM/IEEE conference on*, pages 296–305. IEEE, 2004.

[10] Hui Han, Hongyuan Zha, and C Lee Giles. Name disambiguation in author citations using a k-way spectral clustering method. 2005.

[11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[12] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[13] Victor Francisco Rodriguez-Galiano, Bardan Ghimire, John Rogan, Mario Chica-Olmo, and Juan Pedro Rigol-Sanchez. An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 67:93–104, 2012.

[14] Yan-Min Luo, De-Tian Huang, Pei-Zhong Liu, and Hsuan-Ming Feng. An novel random forests and its application to the classification of mangroves remote sensing image. *Multimedia Tools and Applications*, 75(16):9707–9722, 2016.

[15] Pucktada Treeratpituk and C Lee Giles. Disambiguating authors in academic publications using random forests. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, pages 39–48. ACM, 2009.

[16] Thais Mayumi Oshiro, Pedro Santoro Perez, and Jose Augusto Baranauskas. How many trees in a random forest? In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 154–168. Springer, 2012.

[17] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[18] Jiliang Tang, Salem Alelyani, and Huan Liu. Feature selection for classification: A review. *Data Classification: Algorithms and Applications*, page 37, 2014.

[19] Lei Yu and Huan Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 856–863, 2003.

[20] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

[21] Jose G Moreno-Torres, Josea Saez, and Francisco Herrera. Study on the impact of partition-induced dataset shift on k-fold cross-validation. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1304–1312, 2012.

[22] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[23] Kevin Markham. Simple guide to confusion matrix terminology. *Data School [online]*, 2014.

[24] Yutaka Sasaki et al. The truth of the f-measure. *Teach Tutor mater*, 1(5), 2007.

[25] Jeff Jenness and J Judson Wynne. Cohens kappa and classification table metrics 2.0: An arcview 3. x extension for accuracy assessment of spatially explicit models. *Open-File Report OF 2005-1363. Flagstaff, AZ: US Geological Survey, Southwest Biological Science Center. 86 p*, 2005.

[26] Javad Zahiri, Joseph Hannon Bozorgmehr, and Ali Masoudi-Nejad. Computational prediction of protein–protein interaction networks: algorithms and resources. *Current genomics*, 14(6):397–414, 2013.

[27] Jerome Euzenat and Pavel Shvaiko. Basic similarity measures. In *Ontology Matching*, pages 85–120. Springer, 2013.

[28] Felix Naumann. Similarity measures. *Information Systems*, 2013.

[29] Tin Huynh, Kiem Hoang, Tien Do, and Duc Huynh. Vietnamese author name disambiguation for integrating publications from heterogeneous sources. In *ACIIDS (1)*, pages 226–235, 2013.

[30] Allen Taylor. *Semantics For Dummies*. John Wiley & Sons, Inc, 111 River St, 2015.

[31] Tomas Ptacek. *Master Thesis Advanced Methods for Sentence Semantic Similarity*. Pilsen, 2012.

[32] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schutze. Introduction to information retrieval cambridge university press, 2008. *Ch*, 20:405–416.

[33] Vimala Balakrishnan and Ethel Lloyd-Yemoh. Stemming and lemmatization: a comparison of retrieval performances. *Lecture Notes on Software Engineering*, 2(3):262, 2014.

[34] Pantulkar Sravanthi and DRB Srinivasu. Semantic similarity between sentences. 2017.

[35] Peter Kolb. Experiments on the difference between semantic similarity and relatedness. pages 81–88, 2009.

[36] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86, 2005.

[37] Agazi Mekonnen and Shamsi Abdullayev. Topic modeling and clustering for analysis of road traffic accidents. *Department of Applied Mechanics, Chalmers University of Technology*, pages 4–64, 2017.

[38] Wilbert Jan Heeringa. *Measuring dialect pronunciation differences using Levenshtein distance*. PhD thesis, University Library Groningen][Host], 2004.

[39] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[40] Jennifer Daniels, Doug Nye, and Gongzhu Hu. Cluster analysis for commonalities between words of different languages. In *Applied Computing and Information Technology*, pages 71–84. Springer, 2016.

[41] Manop Phankokkruad. Efficient similarity measurement by the combination of distance algorithms to identify the duplication relativity. In *International Conference on Computer and Information Science*, pages 219–232. Springer, 2017.

[42] Wael H Gomaa and Aly A Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13), 2013.

[43] Jure Leskovec, Anand Rajaraman, and Jeffrey D Ullman. Mining of massive datasets, 2014.

[44] Baoli Li and Liping Han. Distance weighted cosine similarity measure for text classification. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 611–618. Springer, 2013.

[45] Abinash Tripathy, Ankit Agrawal, and Santanu Kumar Rath. Classification of sentiment reviews using n-gram machine learning approach. *Expert Systems with Applications*, 57:117–126, 2016.

[46] Abdulla Ali. Textual similarity. *Technical University of Denmark*, 2011.

[47] S Elliott. Survey of author name disambiguation: 2004 to 2010 [j/ol]. *Library Philosophy & Practice. http://digitalcommons. unl. edu/cgi/viewcontent. cgi*, 2015.

[48] Hung Nghiep Tran, Tin Huynh, and Tien Do. Author name disambiguation by using deep neural network. In *Asian Conference on Intelligent Information and Database Systems*, pages 123–132. Springer, 2014.

[49] Andreas Strotmann and Dangzhi Zhao. Author name disambiguation: What difference does it make in author-based citation analysis? *Journal of the Association for Information Science and Technology*, 63(9):1820–1833, 2012.

[50] Duncan M McRae-Spencer and Nigel R Shadbolt. Also by the same author: Aktiveauthor, a citation graph approach to name disambiguation. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 53–54. ACM, 2006.

[51] Anderson A Ferreira, Marcos André Gonçalves, and Alberto HF Laender. A brief survey of automatic methods for author name disambiguation. *SIGMOD Record*, 41(2):15, 2012.

[52] Kyohei Atarashi, Satoshi Oyama, Masahito Kurihara, and Kazune Furudo. A deep neural network for pairwise classification: Enabling feature conjunctions and ensuring symmetry. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 83–95. Springer, 2017.

[53] Bart Thijs, Koenraad Debackere, and Wolfgang Glänzel. Improved author profiling through the use of citation classes. *Scientometrics*, 111(2):829–839, 2017.

[54] Denilson Alves Pereira, Berthier Ribeiro-Neto, Nivio Ziviani, Alberto HF Laender, Marcos André Gonçalves, and Anderson A Ferreira. Using web information for author name disambiguation. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, pages 49–58. ACM, 2009.

[55] Ibrahim M. Qdemat. A heuristic-based approach for author name disambiguation. pages 2–19, 2017.

[56] Shahd Zeiad Ewawi. A rule-based clustering method for author name disambiguation in the domain of publications. pages 2–25, January, 2017.

[57] Jerome P Kassirer and Marcia Angell. Redundant publication: A reminder, 1995.