भारतीय सूचना प्रौद्योगिकी संस्थान इलाहाबाद

RWTH AACHEN UNIVERSITY

Final End-Semester Project Report

Department of Information Technology, IIIT Allahabad and
Lehrstühl für Informatik 5, RWTH Aachen University

Bachelor Thesis

B.Tech in Information Technology

Invidenti: Author Disambiguation for Medical Patents

Sanchit Alekh

Advisor(IIIT Allahabad):	Prof. Dr. Uma Shanker Tiwary
Advisor(RWTH Aachen):	PD Dr. Christoph Quix
Roll Number.:	IIT2012108
Address:	Junkerstraße 66, Aachen 52064, Germany
Email:	iit2012108@iiita.ac.in / alekh@dbis.rwth-aachen.de
Telephone:	+91 8601751323 / +49 151 71235742

Date of Submission: **29. July 2016**

Abstract

This Bachelor Thesis work describes and improves on an automatic approach for inventor disambiguation developed by the Databases and Information Systems group at the Lehrstühl für Informatik 5, RWTH Aachen, as a part of the ongoing Mi-Mappa project. The approach makes use of text-mining, logistic regression, an assortment of clustering algorithms and patent-publication matching technique. It makes use of the textual information from patents and their metadata to provide a reliable and powerful method to disambiguate the inventors. This thesis work also aims to improve the performance of the disambiguation method by extending it to multiprocessor architecture and to ensure that the final software follows modern software engineering principles, and is modular, configurable and maintainable.

Certificate

This is to certify that the project titled **Author Disambiguation and Ontology Mapping of Medical Patents** is a bonafide record of the work done by **Sanchit Alekh (IIT2012108)** in partial fulfullment of the requirements of the award of the degree of **Bachelor of Technology** in **Information Technology** at the **Indian Institute of Information Technology Allahabad** during the year 2012-2016.

This work was carried out during his internship period from **January 2016** to **June 2016** at **Lehrstühl für Informatik 5, RWTH Aachen University** under the co-mentorship of **PD. Dr. Christoph Quix**

Allahabad, 29 July 2016

.....

(Advisor - Prof. Uma Shanker Tiwary)

Acknowledgement

I would like to thank PD Dr. Christoph Quix and Prof. Dr. U.S. Tiwary for believing in me and providing constant support during my Bachelor Thesis. Their guidance and suggestions were invaluable for me. I would also like to thank my parents for providing me much-needed mental support at difficult times. The financial assistance provided by Lehrstühl für Informatik 5, RWTH Aachen during the course of the thesis is also heartily acknowledged.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Patents are an excellent embodiment of innovation in several research fields, and information derived from patents has been immensely useful for predicting technology trends, identifying loopholes and vacuums, and keeping track of the technological advancement being pursued by your competitors etc. Overall, patents have become an indispensable part of modern scientific research. In the ever-growing field of medical science however, patents offer an additional advantage. Patent text and metadata can be used to cross-reference and assign patent inventors and collaborators to competence fields, and this can be used to create groups of researchers and scientists to pursue complex innovation in medical science.

An explosion in the number of patents in the recent years has made it extremely difficult to manually analyze, disambiguate and assign them to competence fields. Therefore, automatic approaches have been developed in computer science to aid these processes, and these approaches fall into two major categories : Text-mining techniques and Visualization techniques. While the text-mining techniques focus on natural language processing of the contents and metadata, visualization techniques take the help of patent networks, maps and graphs for a visual form of patent analysis. In both these techniques, inventor identification or disambiguation is a challenging tasks because of a multitude of reasons. Although name is the most intuitive way of identifying an inventor, it has severe limitations. Patent offices such as United States Patent and Trademark Office (USPTO) and the European Patent Office (EPO) etc. don't require patent applicants to format their names in a specific manner. As a result, the same inventor, for example, might be mentioned as 'Mark Jones Smith' in one patent and 'Mark J Smith' in another. Moreover it is difficult to confirm that 'Mark Jones Smith' and 'Mark J Smith' are the same person just by looking at their names in the patent database. Besides, spelling errors in the patent database introduce further ambiguity in inventor identification. The second reason is that two inventors with very different (or same) competence fields may share the same name, and therefore, we have to look towards other features from the patent database to disambiguate the inventors. With the increase in volume of patents,

these occurrences will be more pronounced, making manual or simple name-based disambiguation unreliable.

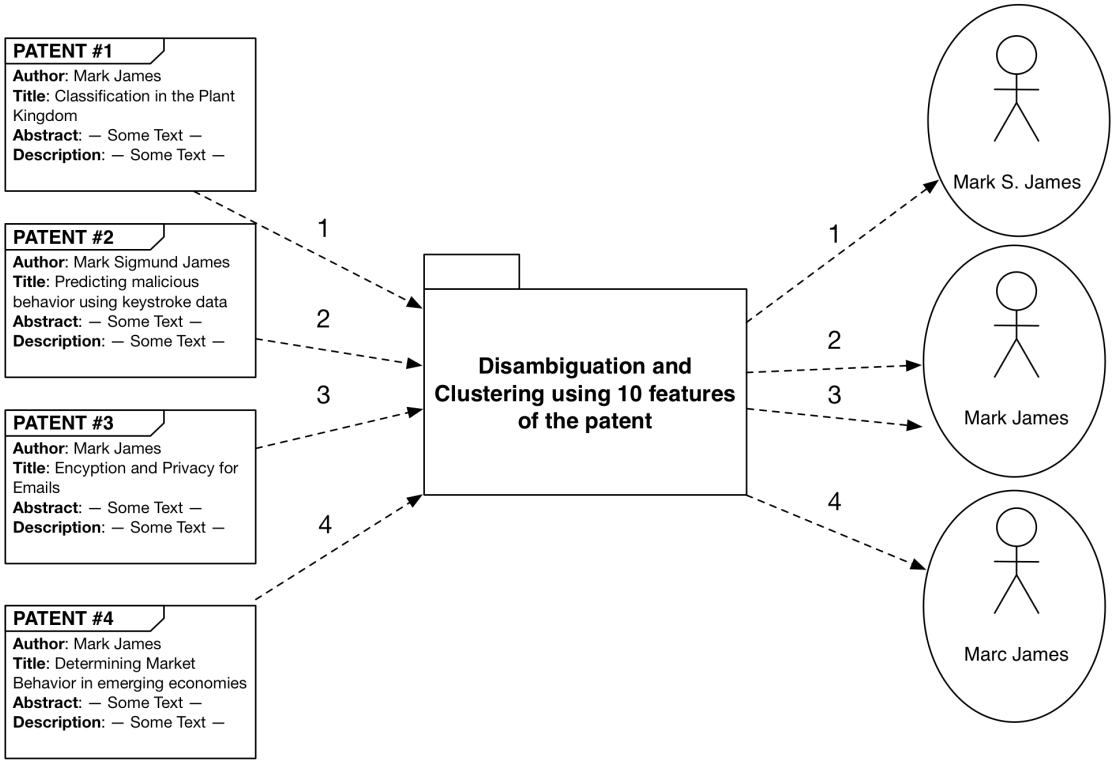

Figure 1.1: Significance of Inventor Disambiguation for Medical Patents

1.2 The mi-Mappa Project

Complex innovations in medical engineering are not possible without collaborative cooperations today. However, the assembly of suitable experts is usually left to the initiating innovators themselves or fortuity. To tackle this problem a new integrative competence model of medical engineering based on data mining algorithms has been conceptualized by the Institute of Applied Medical Engineering (AME), RWTH Aachen. It identifies suitable competence-fields and proposes actors based on published texts for a given project by matching experts from medical, technological and product-related fields to the project. Specifically, the product-related dimension of the approach faces the problem of the correct assignment of patents (and the corresponding inventors) to designated competence fields in medical engineering. The mi-Mappa project attempts

to tackle this challenge by two different but complementary ways: on the one hand a relation between information from medical products and patents is searched for, because medical products are easily assignable to competence fields and hence, the related patents are assignable to competence fields. On the other hand, the project also aims to find publications of the patent innovators related to the project topics which are more easily assignable to competence fields than the patents themselves. The project is based on technologies and knowledge from the fields of ontology modeling and matching, data and text mining and data cleaning. [1]

1.3 Goals

This goals that this inventor disambiguation algorithm plans to achieve can be categorized into two main distinctions: *Functional goals*, that describe the desired functionality provided by the software, and *Performance goals*, which describe the desired performance attributes of the software. Both of these goals are fundamental to this thesis.

1.3.1 Functional Goals

1. **Feature Selection:** For any machine-learning task, finding good features are a prerequisite. The foremost goal of our project is to carefully choose ample number of features which can be easily extracted from the patent document and database, and at the same time, are good enough to disambiguate the inventors. In chapter 1, we have looked at how inventor name is an intuitive but incomplete for inventor disambiguation. Therefore, additional information such as location, assignee, text abstract etc also need to be considered. After the features are selected, the data structures for representing each feature and for aggregating them need to be designed.

2. **Similarity Calculation:** Due to heterogeneity of features, different similarity calculation schemes have to be designed for each of them. For example, in this thesis, Levenshtein Distance is used [2] as a similarity measure between strings, Geographical distance for determining similarity between latitudes and longitudes retrieved from the patent database, and Cosine Similarity [3] for text similarity.

3. **Identify the significance of difference features:** Although a multitude of features are employed for the task, not all the features have the same importance. Some features, eg. name of the inventor might be more useful for classification

than features such as technology class. To reflect this in this thesis, a weighted sum of similarity scores is used to arrive at a global similarity score, which is then matched to a threshold. These weights are calculated via *Logistic Regression*.

4. **Clustering of Patents:** Clustering of the patents is also an important step in this thesis. The clustering algorithm tries to group together patents from the same inventor. In general, clustering algorithms require a pre-defined parameter, such as k-value for k-means clustering. For multi-dimensional data, the clustering algorithms sets a weight for each feature. In this thesis, these weights are the same as the ones from Logistic Regression. In addition, a transitivity property suggested by Lei [4] is used.

5. **Patent-Publication Matching:** Apart from patent data obtained from the database, publications of the patent author are also important identifying information [4]. In this thesis, patent-publication matching is also used as a complementary method to improve the accuracy of clustering.

1.3.2 Performance Goals

1. **Software Design and Architecture:** As a part of the code-base for a larger project mi-Mappa [1], the software architecture and design for this thesis must conform to modern software-engineering principles to ensure code maintainability and possibility of future extension. Hence, to make the design of the future-proof, it is ensured that the code confirm to *S.O.L.I.D* principles suggested by Martin [5]. They are :

   a) *Single Responsibility Principle:* A class should have one, and only one, reason to change.

   b) *Open Closed Principle:* You should be able to extend a classes behavior, without modifying it.

   c) *Liskov Substitution Principle:* Derived classes must be substitutable for their base classes.

   d) *Interface Segregation Principle:* Make fine grained interfaces that are client specific.

   e) *Dependency Inversion Principle:* Depend on abstractions, not on concretions.

2. **Support for Parallelization and Multiprocessor Architecture:** The input database for this thesis is the USPTO database used by Fleming et al. [6], which is in excess of *5.7 Gigabytes*. To process such a large and rapidly-increasing database, powerful processing is required, which is why the code should be designed from ground-up in such a manner that it supports multiprocessor architectures and parallel processing. In this thesis, *Java Threads* have been used to ensure the same.

3. **Self-Configurability:** Several mathematical and machine-learning approaches are available for performing tasks such as clustering, classification, weight training etc. Some approaches are more useful than others in specific instances.The disambiguation process in this thesis has been designed to incorporate more than a single type of algorithm. Eg. both conventional logistic regression and logistic regression with bold driver method have been implemented. The software should be self-configurable to use the optimal algorithm for the task.

4. **Optimal Use of System Resources:** As a pre-requisite for high-performance computing, the code should be able to make optimal use of system resources. Therefore, the code has to be written in a way that it optimizes memory and CPU usage at every stage of the process.

1.4 Project Documentation

Along with the goal of writing consolidated, clear and scrutable code, our motive from the outset, was to create a complete and lucid documentation for the code-base, including UML class and sequence diagrams, JavaDoc documentation for all classes and methods and a Project wiki to be hosted on a web-based Git repository hosting service. The documentation helps not only in the long-term maintainability of the code, but also aids new programmers to quickly understand and add new features to the project due to its highly extensible project architecture. This also makes the project future-proof, because when new, more efficient algorithms for the current tasks are invented, they can be deployed to the project easily. Project documentation for *InvIdenti* is available in the following forms:

1. **UML Diagrams:** UML, or *Unified Modeling Language* [7] is a very important modeling tool which can help represent a project in a higher abstraction than the code itself. This makes the project structured in a way that enables scalability,

security, robust execution under stressful conditions, and aids maintenance programmers to find and fix a bug that shows up long after the original authors have moved on to other projects [8]. For this project, the *Sequence Diagram* and *Class Diagram* are the most important illustrations.

2. **JavaDoc Documentation:** The Javadoc™tool parses the declarations and documentation comments in a set of Java source files and produces a corresponding set of HTML pages describing the public and protected classes, nested classes, interfaces, constructors, methods, and fields [9]. JavaDoc is the de-facto standard for generating API documentation for Java code, and is very helpful not only for programmers willing to reuse the code, but also for maintenance programmers. In *InvIdenti*, we have tried our best to provide complete and comprehensive API documentation.

3. **Wiki Pages:** While API documentation is a very helpful tool, it is mostly intended for people with a technical background. Wiki Pages, on the other hand, provide more more verbose information about the project, the tools used and details about the implementation. *InvIdenti's* Wiki Documentation is organized under three major heads, *Introduction, Techniques Used* and *Implementation Detail*, and is written in a way that allows even a technically non-inclined person to have a gainful insight on the project.

2 Related Work

Traditionally, the inventor identification task has generally been divided into three steps: 1) data cleaning and parsing, 2) data matching and 3) data disambiguation [10]. While the first two steps try to match the same objects with different representations, the third step is to retain the correct matches and remove the incorrect ones. During the disambiguation process, some additional information is used as reference. This information is classified into two categories: patent information and non-patent information. Information such as location, assignees, co-inventors, names of the inventors etc. count as patent information. These can be extracted from the patent document. Similarities between the inventors of different patents are computed based on these information. For example, the location is used to calculate the geographical distance between two inventors as a similarity measurement. Ideally, for the similarity scores between two inventors who are actually are same person, similarities based on one or more features be high. However, sometimes the similarity scores are also high even for two different inventors, especially for those who usually cooperate with each other [4]. Therefore, the disambiguation method should be smart enough to take these observations into account. The matching of the inventor and author usually uses some methods such as institutional matching, geographical location matching etc [10], [11], while a text-mining based approach has also been introduced by Cassiman [12]. The following subsections in this chapter will introduce some of the novel disambiguation approaches that have been developed so far.

2.1 Identifying Author-Inventors from Spain

Maraut introduced a novel approach for inventor-author identification from Spain [10]. The approach is divided into four steps. The first step is to create a structure of the name and address representations of the patents and the publications. The second step involves matching the inventor and author by using the name and the address. The address of the author is the institution address which the author is affiliated to while the addresses of the inventor are the addresses of the applicants and the inventors. In the third step, a global similarity score is calculated, which can be used for clustering the inventors and authors. The inventors and authors in the same cluster are considered as the same

person. The fourth step is to control the data quality and improve the disambiguation manually by using the recursive methods. This approach finds the weights by observation and experimentation.

2.2 Text-Mining Approaches

A method based on text-mining approach was introduced by Cassiman to match the inventor of a patent and the author of a publication [12]. His approach first extracts the key words of the abstract of the patents and publications respectively, and uses the intersection between the sets of the keywords of the publications and patents as the final *term set*. Then he generates a k-dimensional vector for each of the patent and publication respectively where k is the size of the final term set. An element in the vector is the weight of a term in the document which is computed by the term frequency and inverse document frequency. Then the similarity for each pair of the patent and publication is calculated by using the cosine of the angle between the vectors. Each patent is assigned the n most relevant publications, where n is defined manually. The inventors of the patent and the authors of the related publications are matched if they have the same last name. Cassiman evaluated this approach by setting n to 20 which results in a 66% successful matching. Because text similarities are usually larger than zero, the patents can usually be found to match certain publications although they are from different authors.

2.3 Inventor-Author Matching by Rare Name

Boyack and Klavans [11] introduce an inventor-author matching approach based on rare names. This approach is based on the assumption that if an inventor and an author have the same name and that name is rare, then they should refer to the same person. The approach calculates the rare rate for the names of the inventors and the authors. The rare rate of an author name is calculated as the largest percentage of the publications which belong to a certain affiliation. The rare rate of the inventor name is calculated in the same way but based on the information of the assignees. The inventor and the author are matched if they have the same name and their rare rates are bigger than a predefined threshold. This approach results in a 25% matching rate. The limitation of this approach is that it can only deal with rare names. Therefore, it cannot be applied for a large-scale inventor identification of the entire patent database.

2.4 Torvik-Smalheiser Disambiguation Method

Torvik and Smalheiser introduced a statistical model [13] that predicts, for any two Medline articles sharing the same author, the probability that they are written by the same individual. They used a vector of features such as shared title words, journal name, co-author name, medical subject headings, language, affiliation and the presence of middle name and suffix in the author's name. Their vector method took into account the nonlinear and interactive effects across features; moreover, positive and negative training sets were very large and constructed automatically, which allowed for very robust results. Thus, given any pair of papers bearing the same author, they computed the comparison vector and observed its relative frequency in the positive vs. negative training sets (the r-value). If the observed profile was much more frequent in the positive set than in the negative set, they concluded that it is likely that the two papers were written by the same individual. Using this approach, Torvik and Smalhesiser were able to achieve a recall value of 0.98.

2.5 Fleming's Inventor Disambiguation Method

Improving on the Torvik-Smalheiser Disambiguation Method [13] Fleming et al. developed an approach [6] by using the naive-Bayesian classifier technique for inventor disambiguation. Their approach first selected a subset of the information from the raw patent data as features to represent the patent with a special inventor from the patent-inventor list. They called this special form of patent as an inventor-patent instance. Pairs of the inventor-patent instances were the basic units for the naive Bayesian classifier. A similarity profile which contained all the similarity scores based on different features was calculated and a label was assigned to indicate whether the inventor-patent instances were from the same inventor or not. The naive Bayesian classifier learned the likelihood by using a training dataset. In order to apply it to a large dataset, Fleming used blocking techniques. This approach created blocks of the inventor-patent instances. They used the likelihood for each pair of the inventor-patent instances to do the agglomerative clustering until the log-likelihood reached its maximum level.

3 Solution

3.1 Early Experiments

3.1.1 On the possible use of Bigrams

To inquire into the possible use of Bigrams in the Vector-Space Model, medical words from three standard medical dictionaries were aggregated. On calculating the average word length of a medical term, it came out to be **1.893789**. The closeness of the word-length to 2 made a strong case for the introduction of bigrams in the Vector-Space Model to improve the performance of Text Mining. However, in a preliminary analysis, it was observed that this is not the case, and that bigrams did not lead to an improvement in the text-mining process. Hence, this idea was abandoned.

3.1.2 On the possible use of application-specific Stop-Words

To find out possible candidates for stop-words tailored to medical patents, a 28MB long text file containing the abstracts of 7000 patent documents was aggregated, and the frequently occurring terms in those patents were analyzed. Some interesting results came up. These statistics were surprising, because words such as 'invention', 'method' and 'treat' occured very frequently, and as a healthy fraction of some of the traditional stop-words 'the', 'and' etc. *Table 3.1* lists some of the traditionally used stop-words along with their occurrences in the sample document set, whereas *Table 3.2* lists of the frequently occurring words in the sample document set and their frequencies. This observation made a strong case for the use of domain-specific stop words for the preprocessing task in the text mining process. However, in preliminary analysis, this too, didn't lead to any visible improvement in performance, and therefore, was abandoned.

3.2 Basic Underlying Idea

Each patent usually contains a list of inventors. The basic data unit used for inventor identification in this thesis is an amalgamation of the patent and one of the inventors from the inventor list. For convenience, the basic data unit uses the same name in Fleming's

approach [6], i.e. *inventor-patent instance*. There is a separate inventor-patent instance for each co-author of a patent. Eg. if the patent contains three inventors, there are three inventor-patent instances for it. As suggested by Lei [4] *InvIdenti*, an assortment of 10 features is used to represent each patent-inventor instance. Between different inventor-patent instances, the similarities based on different features are computed and normalized. The weighted sum of these similarities is computed as the global similarity, as shown in *Equation 3.1*.

$$S_{global} = \sum_{i} w_i \cdot S_i \tag{3.1}$$

If the global similarity is larger than a threshold ϵ, the two inventor-patent instances are considered to belong to the same person. The weights represent the importance for the feature similarity. This implementation poses two major challenges, which must be solved in advance. The first is the differences in similarity calculation methods due to heterogeneity of the features. Therefore, different similarity calculation methods must be designed for them. The second challenge is that the weights w and the threshold ϵ should be assigned suitable values using an algorithmic approach, because manual tuning using heuristic indicators is not suitable in this case. In this thesis, logistic regression is used to find the suitable values for weights and threshold by training a inventor-patentinstance dataset. After the logistic regression, the transitivity property, suggested by Lei [4] ,is performed by using two different clustering algorithms, namely, Hierarchical clustering and Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

The principal reason to use clustering algorithms is to try to group the inventor-patent instances from the same inventors together. These clustering algorithms use different mechanisms internally to affect the transitivity properties. Hierarchical Clustering uses a tree-based technique to calculate the similarities between clusters and merge them while DBSCAN uses a parameter called *minPts* to affect transitivity.

After clustering is performed, each cluster is considered to contain patents by a single

Word	**Occurrence**
The	236259
And	174952
A	119769
To	72580
Which	15390

Table 3.1: Occurrences of some common stop-words

inventor. Then a refinement may be performed by using the patent-publication matching [4], which provides an extra layer of credence to the classification results. For each inventor-patent instance cluster, the related publications for all the patents are aggregated from the publication database. The patent-publication matching is based on three different methods to identify the inventor-author linkage. The first method is non-patent reference matching. If the patent refers to some publication whose author has the same name as that of the patent-inventor, the patent and the publication are matched. The second method is the assignee-affiliation matching, i.e. if the patent assignee is the same as publication affiliation and the publication author and the patent inventor have the same name, the patent and the publication are matched. The third method is to calculate the similarity between the abstract of the patent and the abstract of the publication, the patent is matched to the publication with the best text similarity, while the inventor and the author have the same name. After patent-publication matching, the author IDs of the publication database are assigned to the clusters. If two clusters have the same author ID, the clusters are merged. On the completion of refinement, the final result of the inventor identification is obtained.

3.3 The Patent-Inventor Unit

Inventor-patent instance is the basic data unit which is used in this thesis. *Figure 3.1* shows the data structure of the inventor-patent instance. There is a lot of information from a patent document which can be used as features for the disambiguation task. Taking Fleming's feature selection [6] as reference, ten features are chosen to represent the inventor-patent instance. The ten features are classified into two categories, *inventor information features* and *patent information features*. The inventor information features are comprised of *last name, first name* and *location*. On the other hand, the patent information features comprise of *abstract, claims, description, title, technology class, co-inventors* and the *assignee(s)*. The names of the inventors are strings without any

Word	**Occurrence**
Invention	43012
Method	39964
Treat	39388
Prepare	39196
Use	38300

Table 3.2: Occurrences of frequent words from medical patents

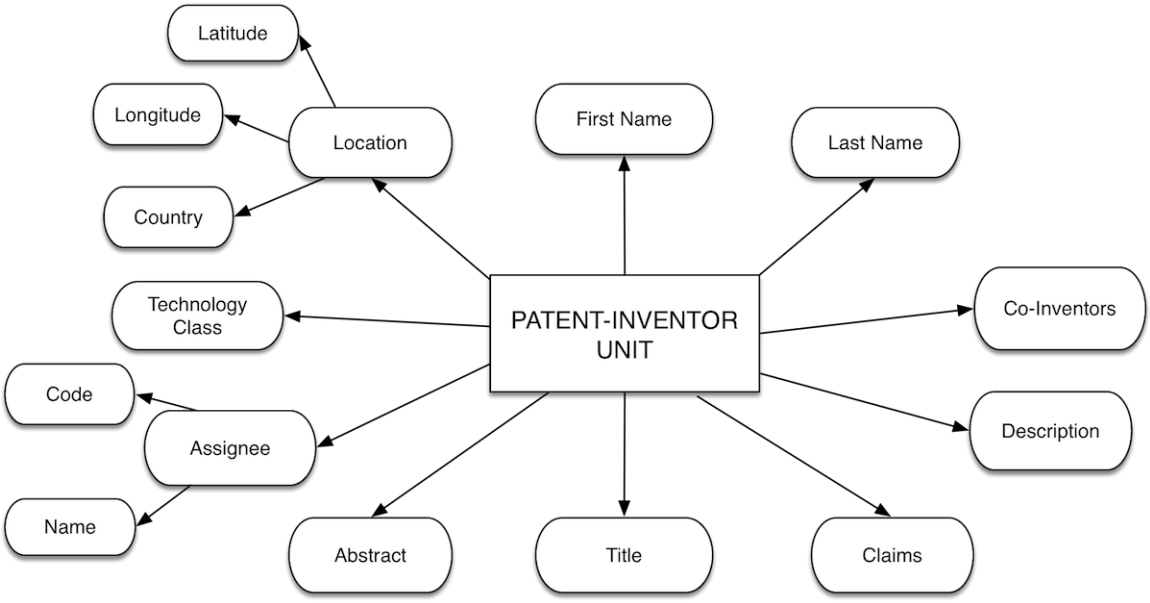

Figure 3.1: The Patent-Inventor Unit

punctuation characters. The case of the names is ignored. Middle names are not available for all inventors, so it was decided not to take it into consideration. The location information contains longitude, latitude and state abbreviation for countries. The title, abstract, claims and description are obtained from the texts of the patent document. The technology class are a list of the numeric code to represent the concurring technology fields of the patent. The technology classes can be divided into sub-class and main-class. Co-inventors are other inventors of the same patent of the inventor-patent instance. The assignee information contains an assign name and an assignee code. This thesis was aided by Fleming's free access to his database of USPTO patents from 1975 to 2010, from which last name, first name, location, technology class, co-inventors and assign information could be easily extracted. The abstract, claim, description and title can be extracted by using the full-text search engine from USPTO. These textual features are also stored as strings.

3.4 Concept Flow of the Inventor Disambiguation Approach

Figure 3.2 shows the flow chart of the approach used in this thesis. For my approach, an inventor-patent instance dataset with correctly-assigned labels is used as the training dataset. The inventor-patent instance dataset is first subjected to preprocessing. The corresponding feature information is extracted from Fleming's database and the texts of

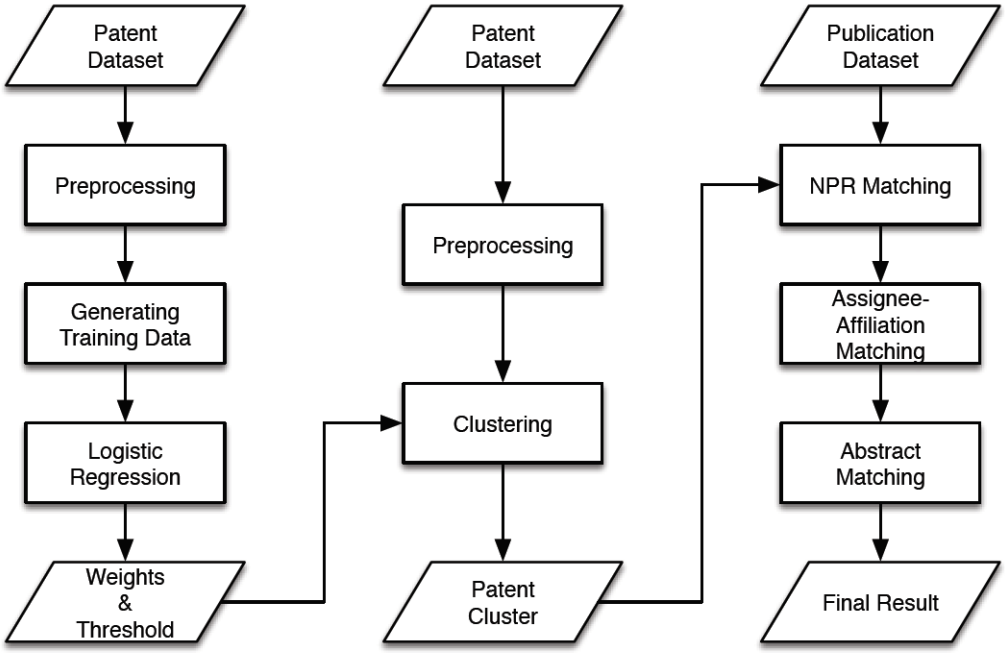

Figure 3.2: The Concept Flow of the Thesis

the patent are extracted by using the patent full-text search engine. After extracting this information, text mining techniques based on the Vector-Space Model are applied on the patent texts, whereby, first, stop-word removal and stemming is performed. Thereafter, term frequency calculation and latent semantic indexing is performed on the patent features. As a result, the patent textual features are transformed into vector representations.

The vector representation is very important because it is used to calculate the similarities between texts. After preprocessing, the inventor-patent instances are used to generate a training dataset for the logistic regression phase. The training dataset contains all the pairs of the inventor-patent instances. We use a binary logistic regression to train the model. For each pair of inventor-patent instance, their similarity values are calculated, which serve as training data. The training data also contains the label to indicate whether the two inventor-patent instances are from the same author.

The training dataset is used by logistic regression to find the suitable values for the weights and threshold. The weights and the threshold are passed on to the clustering methods. The weights are used to calculate the global similarity of the two inventor-patent instances. The global similarity is the measurement to be used to group the

inventor-patent instances. The threshold is assigned to some pre-defined parameters for the clustering algorithms. Traditionally, clustering techniques depend on pre-defined parameters, for example, k in K-means clustering. The value of this parameter significantly affects the quality of clustering. It is extremely difficult to manually/heuristically find the value of the pre-defined parameters of clustering. However, in this thesis, the computed values obtained from Logistic Regression helps us find the values of these parameters.

3.5 Performance Improvement

The dataset obtained from Fleming's experiment [6] contains a total of 32495 inventor-patent instances. As we shall see in later chapters, the Logistic Regression step would require an exhaustive generation of all pairs of documents, giving rise to a total of $\frac{n(n-1)}{2}$ instances of training data for n training documents. To put it into perspective, if we use 10,000 training documents, we will have to generate and process 49,995,000 training instances. Therefore, the text mining as well as training and clustering processes should be made to be as fast and resource-efficient as possible. In this thesis, the slowest processes that created bottlenecks in the project were analyzed and parallelized. If there are multiple CPUs, multiple threads can execute at the same time. Especially for multi-core CPU architectures, a parallelized approach such as multithreading presents a far better alternative to make full use of the system's resources compared to sequential processing. It also leads to more responsiveness, resource sharing and the possibility to overlap I/O and computation. The project development is done in Java™, which natively supports multithreading through its Thread library [14].

4 Implementation

In the previous chapter, I described the brief concept flow of this thesis. Through the course of this chapter, I will sequentially describe each process in detail. The chapter is organized as follows. First, we list down and describe the different features used to represent the Inventor-Patent instance, and the reasoning behind it. Next, we look at the methods to calculate the similarity values for each of the features. Since the features and their quantification is heterogeneous, each feature has to have a different kind of a similarity measure. Further, we look at the training phase using Logistic Regression, followed by Clustering. We will also have a look at the significance and use of Transitivity in this thesis. We will also have analyze the Patent-Publication process, and finally introduce some fine-grained practical issues encountered during the implementation.

4.1 Feature Selection

In any Machine Learning task, feature selection is at the heart of its effectiveness. Highly discriminating features can make the task very precise, while on the other hand, if features are not carefully handpicked for the task, it might lead to poor performance by the algorithm. Patents provide a unique use-case. Their metadata is as important, if not more, than their texts . Therefore the final feature vector for the disambiguation task must incorporate both textual and metadata features. Moreover, while *First Name* and *Last Name* are the most intuitive discriminating features, assigning them high weights manually is often not a good idea. Indeed, previous implementations of Author disambiguation which have used manual weighing schemes have given poor results. Therefore, an automatic technique for assigning feature weights must be established on the basis of trained values. In Invidenti, this is done via Logistic Regression. The features used in Invidenti are an extention of Fleming's inventor-patent instance. In addition to 6 metadata features, 4 textual features have also been added, which are extracted using a crawler from the USPTO database, completing the modified inventor-patent instance, as shown in *Figure 3.1.* These features have different data types, such as strings, number etc. A detailed description of each of them is provided below :

4.1.1 Inventor Names

The name of an inventor always contains a last name and a first name. Since not every inventor has a middle name, it has been omitted for this approach. The original expression for the inventor names are strings. In order to improve the accuracy, the names of the inventors are preprocessed. The punctuations in the names are removed and all the letters of the name are transformed into capital letters. For each inventor-patent instance, two transformed strings are used to represent the inventor first name and last name.

4.1.2 Patent Assignee

Patent Assignee is the organization which owns the patent. The patents from the same inventor have a high probability to have the same assignees, although it is not universally true. For example, some academic researcher may give the ownership rights of his patents to his research institution or some companies. On the other hand, some bigger companies usually own many patents from different inventors. However, considering that the inventors usually cooperate with a limited number of assignees, assignee information is a good indicator to distinguish between patent inventors. Assignee information contains a string for the assignee name and an assignee code. The assignee code in Fleming's database is from the National Bureau of Economic Research (NBER). Since assignee name does not have a standard form, checking the assignee code is the preferred choice. In case the assignee codes are unavailable, assignee names are used to identify the assignee.

4.1.3 Inventor Location

The patent metadata includes the city, state and country information for each inventor. As inventors usually stay in a specific area, close geographical locations indicated by the metadata can be another indicator to show that they are identical. It?s generally not a good idea to directly compare the city, state and country data, because it?s very difficult to measure the closeness by simple comparison. For example, the cities from two different states sometimes have a smaller distance than the cities from the same state. In Fleming's dataset, the location representation contains latitude, longitude and the country.

4.1.4 Technology Class

Technology class represents the domain/area of research that the patent pertains to. A single patent inventor usually focuses on some fields such as computer science, electronics or quantum physics. Although the inventors may cover several fields, the patents from the same inventors are usually from the same or closely related fields. In addition, inventors with the same name focusing on the same fields is quite rare. Therefore, technology class can be used as an effective differentiator to distinguish between the inventors. USPTO divides the technology classes of the patents into main-classes and sub-classes. The main class contains a list of sub-classes. Although classes usually have some relationship such as the computer science and math, the relationship is difficult to measure. The semantic of the class meaning is ignored for this approach. The classes are represented by numeric codes. The full code meaning along with titles can be found in the USPTO technology class search page[1]. Each main class may contain a list of subclasses if the subclasses are available for the inventor-patent instance.

4.1.5 Co-Inventors

A patent usually contains more than one inventors. According to definition, the inventor-patent instances generated from the same patent contain one inventor from the inventor list of the patent. The other inventors are considered as the co-inventors of the inventor-patent instances. Inventors usually cooperate with their colleagues. The inventors who share a large number of co-inventors have a high probability to be the same person. Co-inventors are stored as a list of strings which represent the inventor names. Our raw inventor-patent data doesn't contain the names of co-inventors, hence a query by using the patent number as a keyword for each inventor-patent instance has to launched. The inventors of the returned inventor-patent instances are used as the co-inventors. Co-inventors is a good indication to help us distinguish the inventors who has the same name, but it is not useful to distinguish between colleagues as they usually share the same co-inventors.

4.1.6 Textual Features: Title, Claims, Abstract and Description

Textual Features- Claims, Abstract, Description and Title: The text of the patent is extracted from the Patent Full-Text Search Engine. From the patent text, we use title, abstract, claim and description as features for disambiguation. The title usually contains

[1] `http://www.uspto.gov/web/patents/classification/selectnumwithtitle.htm`

several words to briefly describe what the patent it is. The abstract is a short paragraph to describe the basic idea of the patent. The claims describe the extent of the patent. The claims are of the utmost importance both during prosecution and litigation alike. The description usually gives the details of the patents. A conventional Vector-Space Model of Text Mining is used for dealing with the text. The texts are transformed into vectors which are easily used to compute the similarities. The text similarities reflect the semantic similarities between patents. The transformation is performed in three stages:

1. Stop Words Removal
2. Stemming, and
3. Term Frequency Calculation

4.2 Similarity Calculation

As we seen before, Similarity Calculation techniques are fundamental to this thesis. Since the features used for classification are heterogeneous, a universal distance/similarity measure would not suffice in our approach. Therefore, we require specialized similarity calculation algorithms for each feature. These algorithms are explained in the following subsections.

4.2.1 String Similarity

Features such as last name, first name, assignee name and inventor names are represented using strings. A simple comparison using full matching of the strings is not precise because of the non-unique forms of the inventor names and spelling errors in the patent database. Eg. the inventor names "HARIOM" and "HARI OM" would be found as dissimilar using this approach, although they might not be. The Levenshtein distance [2] as edit distance of the strings is a good solution. Levenshtein distance uses the smallest number of the operations as the distance to transform one string to another one. These operations include deletion, insertion and substitution. Because most of the strings used in my approach represent names, the different forms of the names are similar to each other. This method performs well to find the similarities of the forms of the strings. Since different strings usually have different length, a normalization should be performed by a division of the number of steps by the maximum length of the string. The range of the normalized Levenshtein distance is $[0, 1]$. In addition, the

Levenshtein distance as a distance function should also be transformed into a similarity value. Therefore, $1 - NormalizedLevenshteinDistance$ is used as the similarity of the strings. Mathematically, the Levenshtein distance between two strings a, b (of length |a| and |b| respectively) is given by $lev_{a,b}(|a|, |b|)$ where

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) \text{ is } 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases} \tag{4.1}$$

where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise, and $lev_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b.

4.2.2 Assignee Similarity

As seen earlier, assignee information for the inventor-patent instance contains two pieces of data: assignee code and assign name, and due to discrepancies in the assignee name, assignee code is the preferred option. Unfortunately, assignee code is not available for every assignee, therefore assignee names have to be used to compute the similarity if at least one of the assignee codes is not available. This is also the reason why the assignee names are kept for the assignee feature data structure. When computing the similarities of two inventor-patent instance assignee feature, we first check the availabilities of their assignee codes. If both of them are available, a simple comparison of the assignee codes is performed. If the assignee codes are same, the assignee similarity is 1, otherwise it is 0. If at least one of the code is not available, then Normalized Levenshtein distance is used to calculate the difference of the assignee names $1 - NormalizedLevenshteinDistance$ is used as the assignee similarity.

4.2.3 Inventor Location Similarity

As we saw in *Section 4.1.3* location information contains the longitude, the latitude and the country information. Geographical distance is used to measure the dissimilarities of the location feature. But since the unit of the distance is meters or kilometers, a transformation must be performed. The location similarity approach used by Fleming [6]

is kept in this thesis. Fleming's location similarity has six levels, where higher level means greater location similarity. If the two inventors are not in the same country, the level is 0. If the two inventors are in the same country, a distance based on the longitude and latitude is calculated. The following formula is used to calculate the level.

$$Level(x) = \begin{cases} 5 & \text{if } x \leq 5km \\ 4 & \text{if } x \leq 10km \\ 3 & \text{if } x \leq 25km \\ 2 & \text{if } x \leq 50km \\ 1 & \text{otherwise} \end{cases} \tag{4.2}$$

After the level value is assigned, it is normalized by dividing it by 5.

4.2.4 Technology Class Similarity

The technology class information of the patents contains a main class and a subclass code list. Each sub class is pre-assigned to one of the main classes. Since some of the patents only contains a main class list, both main class and sub class are used to calculate the similarity between technology classes. Our approach follows the one used by Fleming [6]. The technology class similarity has 4 levels based on how many class codes the inventor-patent instances share. 4 is defined as the maximum similarity value. Normalization is performed by dividing the feature value by 4. The sub-class code of two inventor-patent instances are same only when they are assigned to the same main class. This is because sometimes some sub-classes from different main class have the same name. In addition, some different sub-classes have a close relationship. For example, "Histogram Distribution" and "Probability determination" are two different sub classes but they are similar to each other. However, the relationship between two different classes is difficult to measure. This property of the sub-classes are omitted.

4.2.5 Co-Inventors' Similarity

For each instance, the co-inventor feature contains a list of co-inventor names. The similarity computation of the co-inventors is based on how many co-inventors the two inventor-patent instances share. This approach uses the Fleming's method [6] to calculate co-inventors' similarity. Following is the formula.

$$Level(x) = \begin{cases} 6 & \text{if } x \leq 6 \\ x & \text{if } x \in \{1, 2, 3, 4, 5\} \end{cases} \tag{4.3}$$

The assigned level is normalized by a division of 6. The counting is based on the full-matching of co-inventors name, Levenshtein Distance is not used over here.

4.2.6 Text Similarity

For text-similarity, we use the traditional approach using Vector Spaces. For the textual features Title, Claims, Abstract and Description, normalized term frequency vectors are calculated, where normalized term frequency is defined as

$$V = (v_1, v_2, v_3 ... v_n) \sum_{i=1}^{n} v_i = 1 \tag{4.4}$$

where V is the set of terms

In the vector space model, the similarity between the features becomes is computed as the similarity between the two normalized term frequency vectors in the vector space. There are several techniques to calculate distance/similarity, including *Euclidean Distance*, *Manhattan Distance*, *Cosine Similarity* etc. In this thesis, we use *Cosine Similarity* to compute the distance between the documents represented in the vector space. Cosine Similarity between two document vectors u and v is given by the following equation.

$$(cosine(U, V)) = \frac{\sum_{i=1}^{n} u_i \cdot v_i}{\sqrt{\sum_{i=1}^{n} {u_i}^2} \cdot \sqrt{\sum_{i=1}^{n} {v_i}^2}} \tag{4.5}$$

Figure 4.1 gives a simplified 2D representation of documents in the vector space. The cosine of the angle between the two document vectors gives the Cosine Similarity. When the angle is 0 degrees, it means that the similarity is maximum, i.e. 1, because they are the same vector. Cosine Similarity gives a value between 0 and 1.

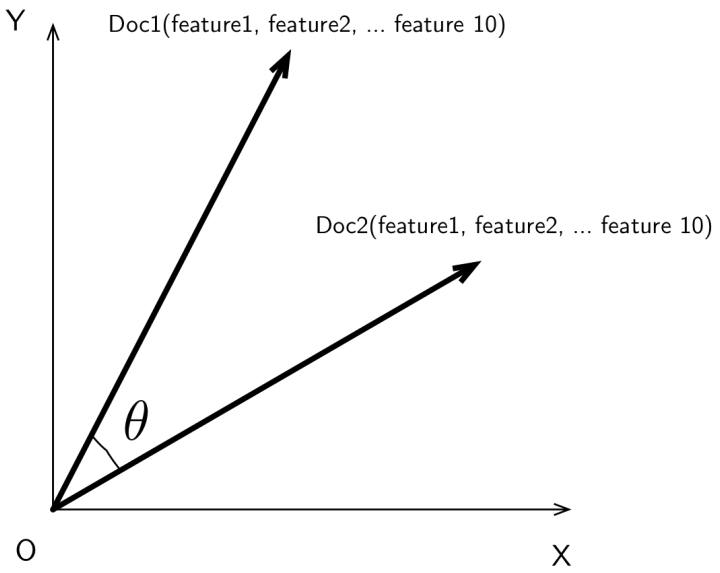

Figure 4.1: Simplified Vector Space Representation of the Patent Documents

4.3 Logistic Regression

Logistic Regression, also known as Logit Regression is a regression model where the dependent variable is categorical in nature, i.e, it can take only fixed values. It is in this respect that it differs from Linear Regression, where the output variable is unbounded in either directions and can take any value. Logistic Regression was developed by statistician *David R. Cox* in 1958 [15]. Logistic Regression as such, in the strictest sense, isn't a classification algorithm, but in Machine Learning tasks, it is often used in this role. Although Logistic Regression can be used to model binary as well as multi-class output variables, binary classification is adequate for inventor identification using our approach.

Logistic Regression is at the heart of our implementation because it trains the training data and gives us the weights of the features used to represent the document vectors as well as the threshold. This has two-fold advantages: (i) It gives us a way to automatically generate the feature weights and threshold from training data rather than using a manual/heuristic approach (ii) As we will see later, these weights and threshold double up as the pre-defined parameters for clustering. In the following sections, we will look at Logistic Regression in a more detailed manner and the gradient-based training method.

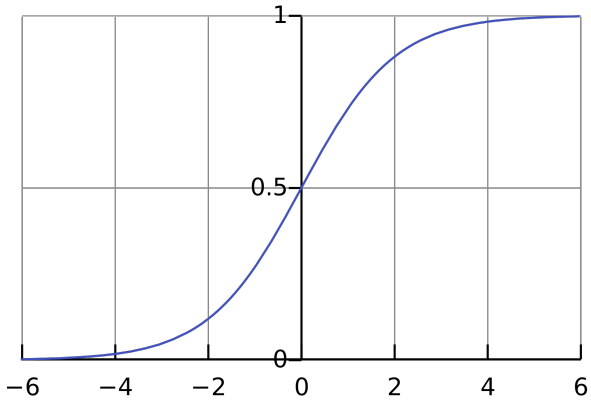

Figure 4.2: The Sigmoid Function

4.3.1 The Logit Model and Logistic Regression

In Logistic Regression, we have a binary output variable $Y \in \{0, 1\}$ and the probability $P(Y = 1|X = x)$ needs to be modeled as a function of the input variable x using maximum likelihood. Since the linear function $p(x)$ is unbounded and the log function $log\ p(x)$ is unbounded only in one direction, the logistic (or logit) transformation is used, which is $log\ (\frac{p}{1-p})$ and we make this a linear function of x. Therefore

$$log\frac{P(Y = 1|X = x)}{1 - P(Y = 1|X = x)} = \sum_{i}^{n} \beta_i \cdot x_i \tag{4.6}$$

In our implementation, the coefficient of linear combination β are the weights of the feature vectors and x are the similarity values corresponding to each feature. Therefore, *Eq. 4.6* can be rewritten as

$$log\frac{P(Y = 1|X = x)}{1 - P(Y = 1|X = x)} = \sum_{i}^{n} x \cdot w^T \tag{4.7}$$

From *Eq. 4.7*, we need to solve for $P(Y = 1|X = x)$ so that we are able to obtain that the inventors from the two inventor-patent instances are a match for the given values of similarity between the features. On solving we get

$$P(Y = 1|X = x) = \frac{1}{1 - e^{-x \cdot w^T}} \tag{4.8}$$

which is the very-familiar looking *Sigmoid or Logistic Function*. As illustrated by *Figure 4.2*, the Sigmoid function is a S-shaped curve which has several desirable properties such as boundedness, differentiability and monotonicity which makes it extremely useful

in diverse fields including artificial neural networks, biology, biomathematics, chemistry, demography, economics, geoscience and statistics. In case of artificial neural networks, for example, the sigmoid function acts as an activation function that transforms linear inputs to nonlinear outputs. Additionally, it gives a bound output to between 0 and 1 so that it can be interpreted as a probability. On the practical side, it also makes computation easier than arbitrary activation functions.

The classical sigmoid function, as seen in *Figure 4.2*, is centred at $(0, \frac{1}{2})$. Its extreme values of 0 (when the input is negatively infinite) and 1 (when the input is positively infinite) makes it desirable to be understood as a probability function, along with its ability to absorb extreme input values without giving wayward output.

With the help of Logistic Regression, our aim is to train the inventor disambiguation model to obtain the weights of the feature vectors and the threshold using a labelled dataset. After this is done, we are able to calculate the weighted sum of the features and compare it to the threshold. If the sum exceeds the threshold, we can say that there is a match using Logistic Regression, and on the other hand, if it falls short, then there is no match. We need to keep in mind, that in order to incorporate the threshold as well as the weights, we need to change $\mathbf{x}$ and $\mathbf{w}$ vectors to reflect the same. Therefore, we use $x = \{1, x_1, x_2, x_3...., x_{10}\}$ and $w = \{w_0, w_1, w_2, w_3, ...w_{10}\}$ where w_0 would be the threshold and $w_1, w_2..., w_{10}$ would be the weights of feature 1 up to 10. After the training is complete, Logistic Regression would be able to assign values to each of the weights as well as the threshold. If the weight of a feature is positive, it can be said that the input x_i is positively correlated to the target value, and on the other hand if the weight of a feature is negative, it can be said that the input x_i is negatively correlated.

4.3.2 Generation of Training Data and Training Process

The inventor-patent instance is illustrated in *Figure 3.1*, and it consists of 6 metadata features and 4 textual features from the patent document. *Table 4.1* enlists a few typical inventor-patent instances (without textual data).

ID	FIRST NAME	LAST NAME	ASSIGNEE	TECH CLASS	LAT	LANG	COUNTRY	CO-INVENTOR
14398723	ALEXANDER V	BORREN	MONSANTO CORP.	381	44.56	-89.52	USA	null
14398723	ALEXANDER V	BORREN	MONSANTO CORP.	381	44.56	-89.52	USA	null
14398723	ALEXANDER V	BORREN	MONSANTO CORP.	381	44.56	-89.52	USA	null

Table 4.1: Example of Inventor-Patent units barring Texts

However, for the training process, these features have to be converted to pair-wise similarity values. In addition, the training instances also have an additional field called

as *'Label'*, which is 1 if the patents belong to the same inventor and 0 if they don't. First of all, pairs of training instances are generated. If there are n inventor-patent instances for training, $(\frac{n(n-1)}{2})$ pairs are generated. To give an estimate of the sheer volume of these document pairs, let us assume that there are $10,000$ inventor-patent instances to be used for training. Under this assumption, $49,995,000$ pairs of documents would have to be generated to calculate the similarity. Hence we also require to use performance optimization techniques, which are described in a separate chapter. Once these pairs are generated, the similarity values are calculated using the techniques introduced in *Section 4.2*. After this transformation step, the dataset is converted to a quantitative representation. *Table 4.3* illustrates an example of a transformed dataset.

LABEL	FIRST NAME	LAST NAME	ASSIGNEE	TECH CLASS	LOCATION	CO-INVENTOR
0	0.15	0.167	0	0	0.2	0

Table 4.2: Example of Training Data

For training, the logistic function must have an associated cost function. The cost function of logistic regression follows a negative logarithmic form, as illustrated by *Equations 4.9 & 4.10*.

$$J(w) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \, log \, h_w(x^{(i)}) + (1 - y^{(i)}) \, log \, (1 - log \, h_w(x^{(i)}))] \tag{4.9}$$

where,

$$h_w(x) = \frac{1}{1 + e^{-x \cdot w^T}} \tag{4.10}$$

Here, x_i is the ith iteration of training data in the training dataset, m is the dataset size and $\frac{1}{m}$ brings the cost function to a normalized form. Cost minimization is performed using *Gradient Descent Algorithm*. The core idea of the Gradient Descent Algorithm is that we start training at a particular point and at each iteration, calculate the error function, and update the parameters until we reach a local minima. In Gradient Descent, the weight update is always proportional to the negative of the gradient of the function at the current point. The gradient of the cost function is given by *Equation 4.11*

$$\frac{\partial J(w)}{\partial w_i} = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - h_w \, (x^{(i)})) \cdot x^i \tag{4.11}$$

Therefore, the weight update of each parameter at every iteration based on the *Gra-*

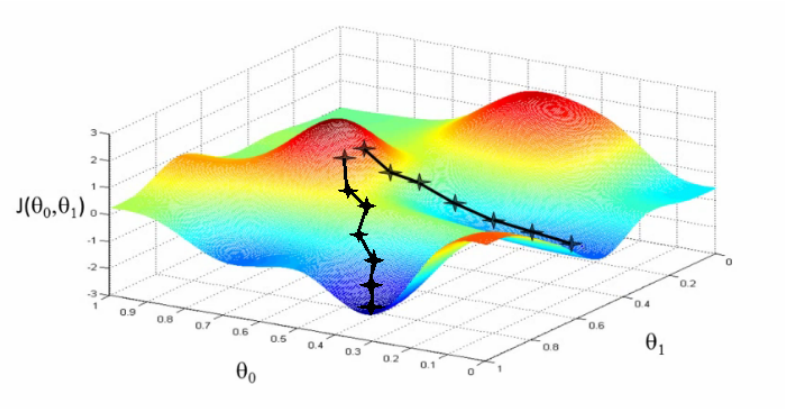

Figure 4.3: Training using Gradient Descent

dient Descent Algorithm is represented as in *Equation 4.12*

$$w_i^{t+1} = w_i^t - \frac{\alpha}{m} \sum_{i=1}^{m} \left(y^{(i)} - h_w\left(x^{(i)}\right)\right) \cdot x^i \tag{4.12}$$

where t is the current iteration of the training process and α is the rate of training, also called as the *step size*. The value of α decides the magnitude by which the weight correction is done in every iteration. It is a very important consideration in the training process because an incorrect selection of α would mean that it would take much longer to reach the minima. Moreover, different values chosen for α decides the manner in which the minima will be reach or if the training will fail. An advantage of using the logit function, however, is that the cost function is convex, i.e., the local minimum is guaranteed to be the global minimum. As shown in *Figure 4.4*, for each training, there exists an α_{opt} that would reach the minima in just one step. If the actual learning rate $\alpha < \alpha_{opt}$ then the training will require more iterations to reach the minima. If $\alpha_{opt} < \alpha < 2\alpha_{opt}$, then the cost function would oscillate in the vicinity of the local minimum.

4.4 Transitivity and Clustering

This approach uses clustering algorithms to accumulate patents with the same author. After the training using logistic regression, we can identify if two inventor-patent instances have the same inventor. However, in reality, many patents have no similarities except the names of the inventor. Sometimes, these inventor-patent instances are considered to belong to different inventors by using logistic regression.

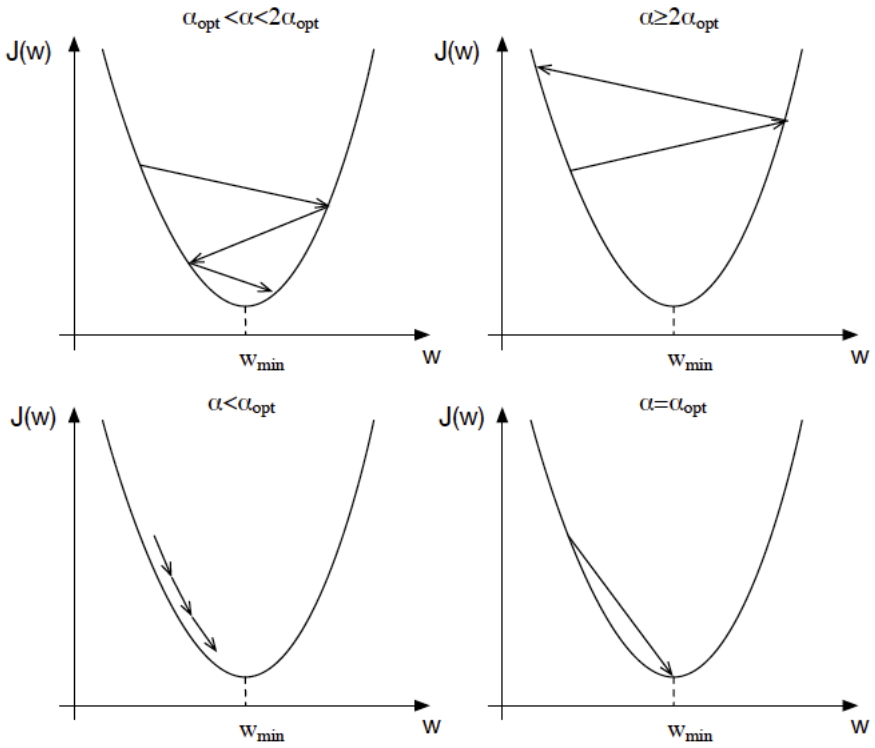

Figure 4.4: Effect of α on the Training

Clustering algorithms aim at helping to solve this problems by adding a transitive property on the identity of the inventor-patent instances to increase the accuracy of the inventor identification process. The clustering algorithm usually needs some distance function or similarity function to calculate the similarity between different objects. If the objects are multi-dimensional in nature, the importance of the dimension is difficult to measure. To remedy this, the clustering algorithm uses the results of the logistic regression, which is very smart way to reuse trained values instead of using manual or heuristic values. Weights are used to calculate the global similarities, and threshold from Logistic Regression is used for the assignment of the pre-defined parameter of the clustering method. The transitivity property might misclassify by considering patents from different inventors to be from the same inventor, and therefore there should be a mechanism to control the transitivity. Clustering algorithms are a good choice for this task. For this approach, the clustering methods attempt to group all inventor-patent instances from the same inventor, as illustrated by *Figure 4.5*

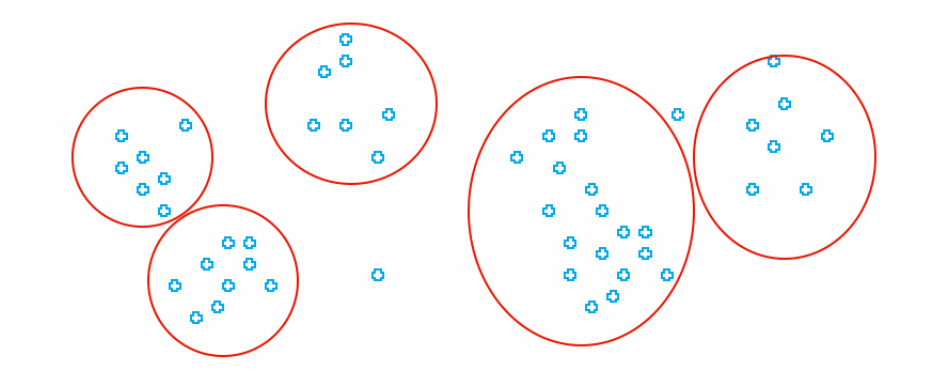

Figure 4.5: Clustering to group inventor-patent instances from same author

4.4.1 The Importance of Transitivity

We use the Logistic Regression method with an assortment of textual and metadata features to train the classifier. However, there are specific instances where the Logistic Regression is doomed to misclassify. A patent inventor might have patents spanning several different fields, for example an expert on Communication Systems can also have patents in the area of Network Security or Wireless Sensor Networks, which are different domains (and different Technology Classes in patent metadata). Moreover, inventors may also change workplaces (hence assignees), geographical location and co-inventors. In that case, patents may have nothing common with each other except the inventor's name (which may also use different forms).

This is where the concept of Transitivity comes into effect. Transitivity is a novel technique used in Invidenti which is suggested by Lei [4], and assumes that the all the patent features do not abruptly change together. As shown in *Figure 4.6* for example, for a seemingly dissimilar patent-pair Patent 1 and Patent 3 with no common features except inventor name, there is a Patent 2 by the same inventor, which has some features in common with both the patents. Therefore, putting it into perspective, if Patent 1 and Patent 2 can be said to belong to the same inventor, and Patent 2 and Patent 3 are said to belong to the same inventor, it can be said with a high degree of conviction that Patent 1 and Patent 3 are also by the same inventor. However, the Transitive property, if applied unchecked, can also misclassify patents from different inventors by saying that they belong to the same one. Clustering is a smart way to ensure that the extent of application of the transitive property is kept into check. For that, clustering uses the Threshold obtained during LR-Training to stop further agglomeration when the similarity metric falls below the threshold.

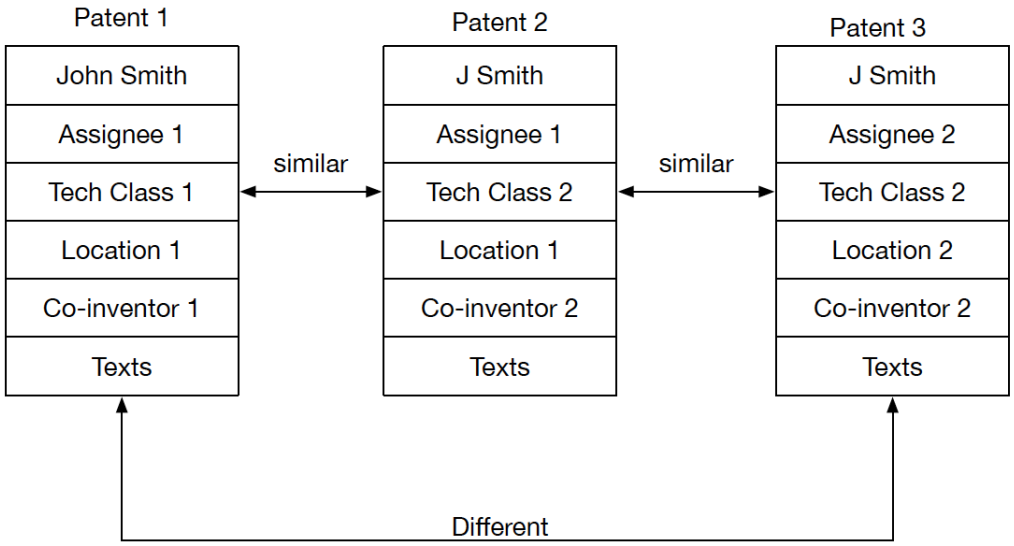

Figure 4.6: Enhanced Matching using Transitivity

4.4.2 Hierarchical Agglomerative Clustering

Agglomerative hierarchical clustering is a classical clustering technique. It merges successively the clusters of objects until all the objects are in the one cluster. At the outset, every object is first put into a single cluster. For each iteration, the similarities between all pairs of the clusters are computed. The two clusters with best similarity metrics are merged into one cluster. We keep doing that until all the objects are in the same cluster. The objects in our approach are inventor-patent instances while their similarity values are the global similarities as weight sums of all the feature similarities. For our approach, the target of the clustering algorithm is to put all the inventor-patent instances of the same inventor into the same cluster and therefore, it is not necessary to keep the hierarchical clustering iterating until only one cluster is left. The clustering process would be stopped based on a suitable criterion. The stopping criterion for the hierarchical clustering is the threshold from the logistic regression training result. If in some iteration, the best similarity of the clusters is less than the threshold obtained from Logistic Regression, which means the patent clusters has a small probability from the same inventor, then the clustering method is stopped.

Algorithm 1 : Agglomerative Hierarchical Clustering:

Input: a set of inventor-patent instances $\mathbf{X} = \{x_1, x_2, x_3...., x_n\}$

Output: a set of clusters $\mathbf{C} = \{c_1, c_2, c_3...., c_m\}$

1: **for** $i = 1$ to n **do**
2: $\quad c_i = x_i$
3: **end for**
4: **do**
5: $\quad (C_i, C_j) =$ maximum sim (c_i, c_j) for all c_i, c_j in C
6: $\quad$ remove C_i, C_j from C
7: $\quad$ add (C_i, C_j) in C
8: $\quad maxSim = sim(C_i, C_j)$
9: **while** $(C.size > 1$ and $maxSim > \xi)$

4.4.3 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) finds core objects, that is, objects that have dense neighborhoods. It connects core objects and their neighborhoods to form dense regions as clusters. DBSCAN was developed by Ester, Kriegel, Sander and Xu at the Ludwig-Maximilians-Universität in München. A user-specified parameter $\xi > 0$ is used to specify the radius of a neighborhood we consider for every object.

The ξ-neighborhood of an object o is the space within a radius centered at o. Due to the fixed neighborhood size parameterized by , the density of a neighborhood can be measured simply by the number of objects in the neighborhood. To determine whether a neighborhood is dense or not, DBSCAN uses another user-specified parameter, MinPts,

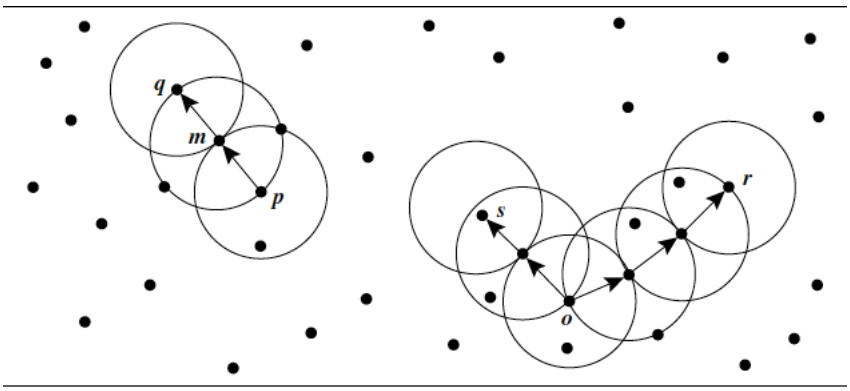

Figure 4.7: Cluster formation in DBSCAN using density-connectedness

which specifies the density threshold of dense regions. An object is a core object if the -neighborhood of the object contains at least MinPts objects. Core objects are the pillars of dense regions. In essence, DBScan tries to build clusters by successively finding all density reachable objects of the core objects and assign them to clusters.

Algorithm 2 : DBSCAN: a density-based clustering algorithm

Input: *D :* a data set containing n objects
ξ *:* the radius parameter, and
MinPts : the neighborhood density threshold

Output: A set of density based clusters

```
1: mark all objects as unvisited
2: do
3:     randomly select an unvisited node p;
4:     mark p as visited
5:     if the ξ neighborhood of p has at least MinPts objects then
6:         create a new cluster C, and add p to C;
7:         let N be the set of objects in the ξ-neighborhood of p;
8:         for each point p^1 in N do
9:             if p^1 is unvisited then;
10:                mark p^1 as visited;
11:                if the ξ neighborhood of p^1 has at least MinPts points,
12:                add those points to N;
13:            end if
14:            if p^1 is not yet a member of any cluster, add p^1 to C;
15:        end for
16:        output C;
17:    else mark p as noise;
18:    end if
19: while no object is unvisited;
```

The classical DBSCAN algorithms usually uses a distance function to measure the difference of two objects. For InvIdenti however, the similarity function obtained from the training is reused. The neighbors of an inventor-patent instance o are the instances which have a similarity larger than a specified value with o. The threshold learnt by the logistic regression is used as the specified value here. A similarity larger than a threshold implies the two inventor-patent instances from the same person, hence it's obvious that inventor-patent instances from the same person are neighbors of each other. In fact, DBSCAN inherently uses transitivity to collect all the objects and assign them to the same cluster. Transitivity is represented by the density-connected concept. Two objects

are density connected if they are density reachable from a core object. The core object is used to connect objects and the core objects are defined by the number of the *minPts*. Therefore, *minPts* is used to control the transitivity of DBScan. For *InvIdenti*, the neighbour of an object doesn't contain itself. So if the *minPts* is set as 1, DBSCAN will consider all the inventor-patent instances which have at least one similar inventor-patent instance as core objects. This works the same as single-linkage clustering which gives the largest transitivity. Increasing the value of the *minPts* would decrease the identity transitivity. When *minPts* is larger than a particular value, there will be no core objects and all the objects in the dataset are considered as noise.

4.5 Performance and Quality Issues in InvIdenti

The project InvIdenti requires a huge amount of processing, both in the training and testing phases. In the Training phase, the Singular Value Decomposition while doing preprocessing takes up a large portion of the CPU time. On the other hand, in the Testing phase, the calculation of Similarity Matrix for clustering takes up a substantial chunk of time. Therefore, the optimization of these processes, among others would lead to much faster processing. One of the most dreaded problems in Machine Learning is that of 'overfitting', and it must be avoided in Logistic Regression. In addition, the learning rate setting and error correction should be done in an optimal way. These performance and quality issues are discussed in this section.

4.5.1 Multithreading of Latent Semantic Indexing

Latent Semantic Indexing, abbreviated as *LSI* is an important tool for dimensionality-reduction in InvIdenti. Since patents are usually long documents and have a large number of stems, the length of the vector increases, which makes the text-similarity calculation very time-consuming. Fall et al. [16] in their research *Automatic Categorization in International Patent Classification* found out that using the first 300 words from abstract, claims and description sections gives a much improved performance rather than using full-texts no matter which classifiers are used. Therefore, we select the first 300 words from abstract, claims and description for the normalized term-matrix calculation. After that, LSI is performed with an n value of 300, i.e., the eigenvectors with the largest 300 eigenvalues are selected.

In the performance testing, it was found that the single-threaded implementation of LSI was slow, but at the same time, 68-76% of the CPU remained idle, clearly creating

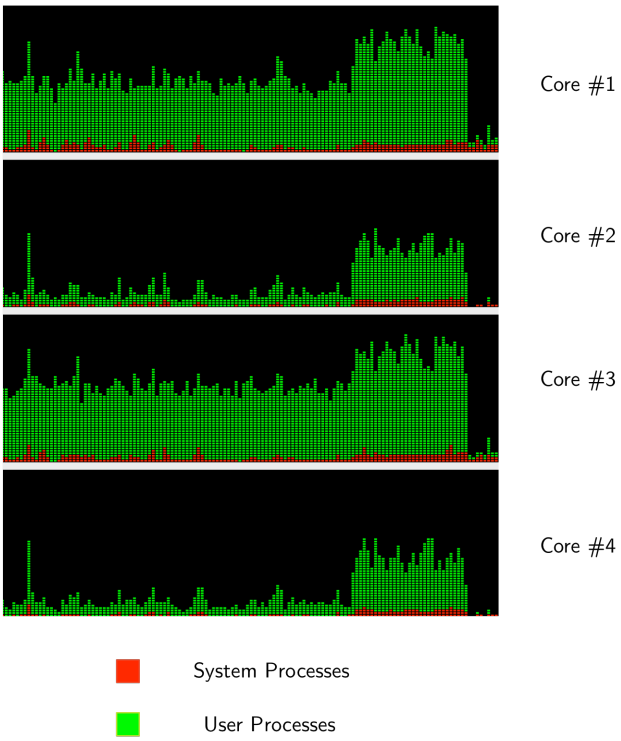

Figure 4.8: CPU Usage of Single-Threaded Latent Semantic Analysis

a deadlock where it neither uses system resources fully nor lets the other processes use it. *Figure 4.8* is a snapshot of the CPU usage of the single-threaded LSI. Therefore, there was a need to make the LSI multi-threaded in order to maximize resource usage, resource sharing and better responsiveness.

The Java platform is designed from the ground up to support concurrent programming, with basic concurrency support in the Java programming language and the Java class libraries [17]. Therefore, the multithreaded application for LSI was developed using native java threads, enabling the LSI for each textual feature, i.e. *Title, Abstract, Claims and Description* to run on a separate thread. The multi-threaded application also makes full of multi-processor architecture which is commonplace in modern computers.

4.5.2 Avoidance of Overfitting in Logistic Regression

Overfitting occurs when a statistical model is not able to successfully learn the underlying relationships in training data and generalize it on untrained data. Instead, it starts to

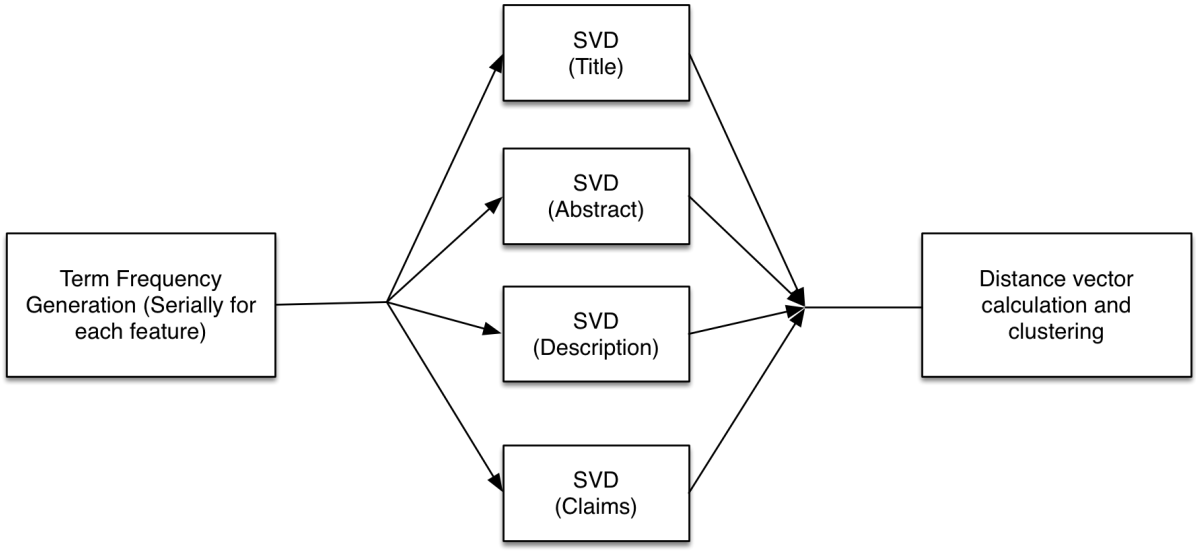

Figure 4.9: Architecture of Parallelized Singular Value Decomposition

memorize the training data and overreacts to minor fluctuations in the training data. Overfitting may lead to poor classification, especially for a project like *InvIdenti* with a large testing dataset. Therefore, avoidance of overfitting is a major concern.

In our test cases, we found that the performance of logistic regression starts monotonically decreasing after a point when training is continued for a long time. There are two widely used techniques, Regularization and *'Stop Early Technique'* to reduce overfitting of sampling data. After adding a regularization parameter, the cost function becomes

$$J(w) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \, log \, h_w(x^{(i)}) + (1 - y^{(i)}) \, log \, (1 - log \, h_w(x^{(i)})) + \lambda R(\theta)] \qquad (4.13)$$

To find the value of the regularization parameter λ, a lot of experiments need to be performed on the training data and its value changes for every new training dataset. This makes it difficult to be incorporated in *InvIdenti*. Instead, the stop-early technique gives a plausible solution without the need to introduce a new variable. In the stop-early technique, we divide the training data into two parts - a training dataset and a validation dataset. The training process uses only the training dataset, and at every iteration, the error from the training dataset as well as the validation dataset is kept in memory. In general, the validation error keeps decreasing up to a point, after which it starts

increasing. The training is stopped at that instant, and the values of the parameters obtained from the previous iteration are used as final values.

4.5.3 Selection of Learning Rate for Logistic Regression

As discussed in *Section 4.3.2*, the value of learning rate α is very important because it determines how fast we will be able to train the model and sometimes it also determines whether or not the training will be successful. With a small α, it takes a long times in order to converge, and on the other hand, a large α might lead to divergence. We found that keeping a constant α led to long training times, and therefore using an adaptive value of α would be the way forward. For this, we use the *Bold Driver Technique* in Logistic Regression. The basic idea behind this technique is that if in the current iteration, we are far away from the global minimum point, the learning rate should be large in order to reach the global minimum fast. If the position is near the global minimum point, then the learning rate should be small so that we don't go over the global minimum point. The Bold Driver technique keeps track of the average training dataset error during the training process. If the average training dataset error decreases, the learning rate should be multiplied by a factor $\beta > 1$. If the average training dataset error increases (meaning that the local minimum point is skipped), the last update of the weights will be cancelled and the learning rate is be multiplied by another factor $0 < \gamma < 1$. The learning rate will be kept decreasing until the training error starts decreasing. Sometimes the learning rate tends to zero, because the current position is near to the global minimum point. Hence, if the learning rate is less than 10^{-10}, the training process will be stopped.

This method has shown in the form of experimental results that it works better than keeping a fixed learning rate in attaining a much faster convergence. For our approach, the learning rate will be given a small learning rate at first. The learning rate is adapted by the Bold Driver technique for each iteration, with the chosen β and γ values 1.1 and 0.5 respectively, in consonance with the recommendations of Battitti [18]. The performance of the training doesn't depend critically on these values.

4.5.4 Parallelization of SimMatrix class

A considerable amount of effort was spent on the parallelization of the *SimMatrix* class. Since the *SimMatrix* class, which calculates the SimilarityMatrix for the clustering, is used by both DBSCAN and Hierarchical Clustering, parallelizing it would mean that the program would run faster while using either of the approaches. The first approach used

was a parallel pool of worker threads, where the number of thread pools was fixed, and determined at runtime by the number of logical processors available for computation the performance. This implementation was not successful and it kept erupting errors due to inconsistencies and race conditions.

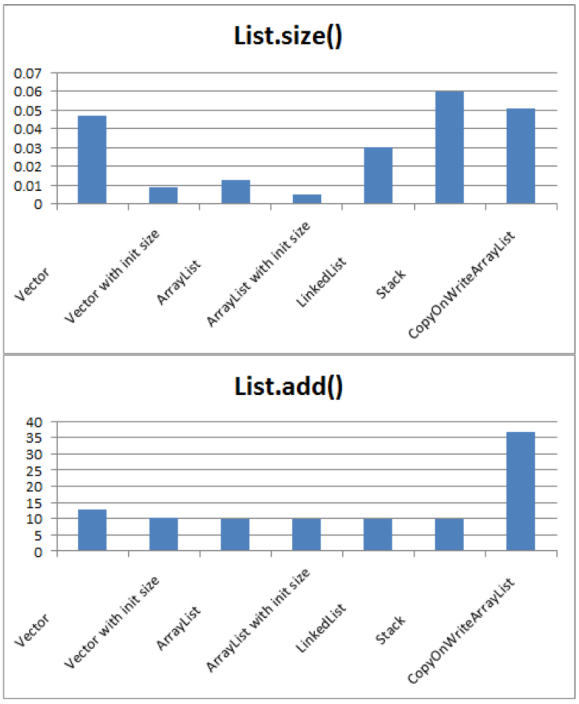

Figure 4.10: CPU Time for *add* and *size* methods for different lists

Although the application ran without producing errors, the performance deteriorated as the number of training/testing patents were increased, making the program unresponsive for as few as 4000 patents, therefore making it unusable. For the entire codebase of *InvIdenti*, the data structure used is an ArrayList, which is known to not be thread-safe. Therefore, parallel modifications using threads on an ArrayList typically leads to inconsistencies and race conditions. The next approach used was a thread-safe variant of ArrayList called CopyOnWriteArrayList in the entire scope of SimMatrix, which solved the problem and the program no longer erupted exceptions. However, the write-times for the CopyOnWriteArrayList are abysmal, as shown in *Figure 4.9* and using a CopyOnWriteArrayList would have reaped benefits only if there were a large number of read operations but very less write operations, which is not the case in our implementation. Therefore, this approach did not work as well.

The third approach was to use cyclic barriers along with pool threads, keeping the data structure as ArrayLists. Cyclic Barriers ensure that until a cycle of worker threads has finished execution, the next cycle does not start. The use of cyclic barriers effectively eliminated race conditions, and the implementation was successful.

5 Java Project Organization

In this chapter, we introduce the more fine-grained implementation details of the Java Project from several abstractions. The basic structure of implementation of the Java project and the flow is introduced first. Then the tools used for development, e.g. development environment, the data structures used for the project and the use of the configuration file (.ini file) is also introduced. In the end, the details of the core parts of the Java project are introduced, along with their UML diagrams. They are related to text extraction, preprocessing, training (logistic regression), clustering, patent-publication matching and evaluation.

Figure 5.1 shows an abstract picture of this thesis, and it is taken from Lei's work [4] on the project. The thesis makes use of two inventor-patent instance databases. The first database is the Fleming's database and the second one is "E&S". Fleming's database is downloaded from their project website hosted on the Harvard University network. [1]The E&S database is created by using the CSV file provided by USPTO [2]. The two main APIs used by the project are the patent full-text search engine API and the other is the Europe PMC API. The first is used to extract the patent texts and the second is used for extracting the related publication information.

The project implementation is divided in 6 phases, shown by rounded rectangles in *Figure 5.1*. The first one is called Text Extractor. It's a spider designed speficially for the patent document file returned from the API PatFt. It is able to extract the title, abstract, claim and description from the patent document, and store the extracted texts to separate text files. The second part is the preprocessing, in which the texts and the data from the database are modified before the training takes place. The third part is the logistic regression. The logistic regression is connected to the two databases and the text dataset generated by the spider. Logistic Regression is used to train the model and find suitable values for threshold and weights to be used further for clustering. In the clustering phase, the clustering algorithms are used to cluster the inventor-patent instances and pass the resulting cluster information to the patent-publication matching part. The patent-publication matching phase deals with the Europe PMC API and tries

[1]**https://dataverse.harvard.edu/dataset.xhtml?persistentId=hdl:1902.1/15705**
[2]**http://www.patentsview.org/workshop/participants.html**

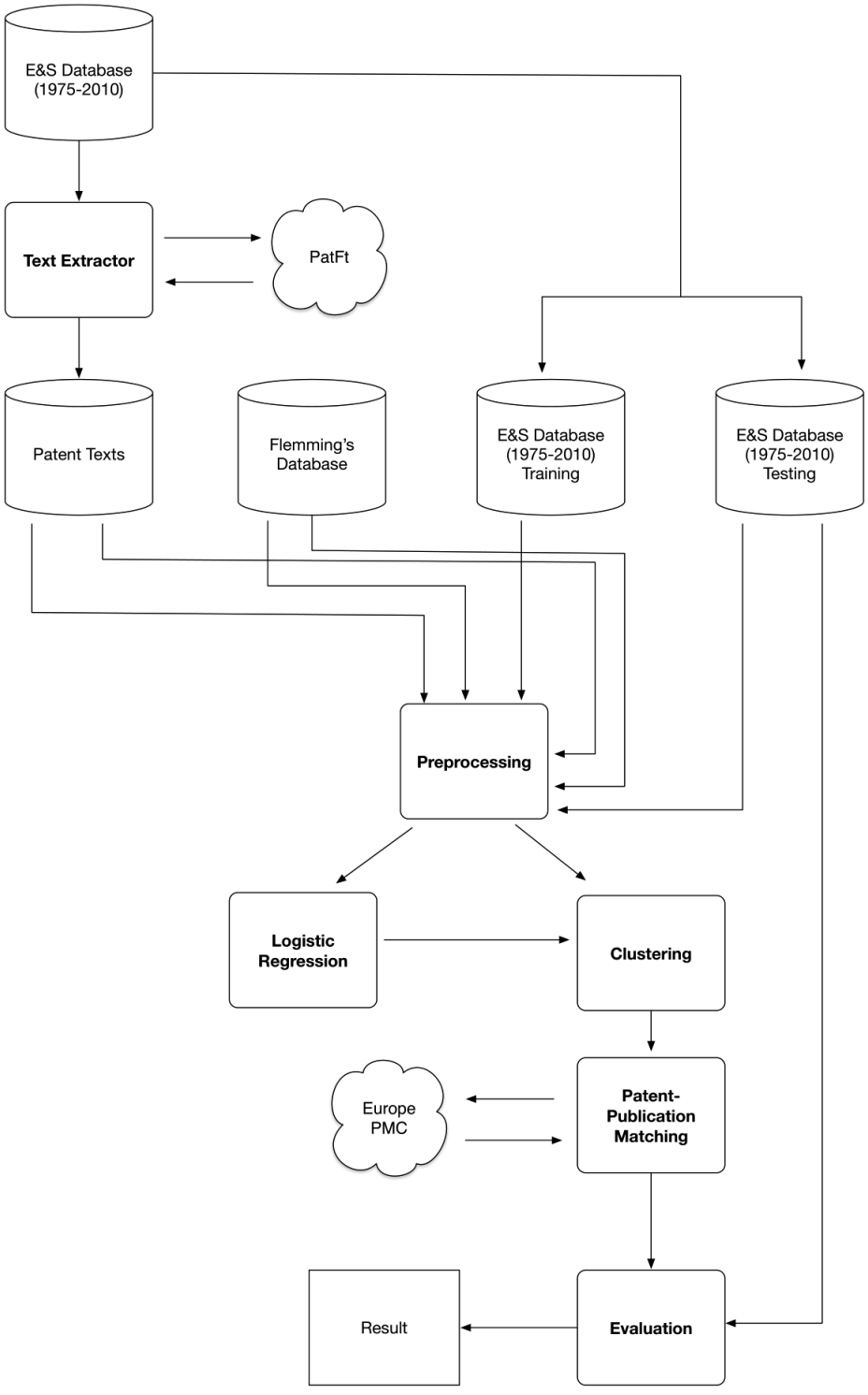

Figure 5.1: Basic Structure of the Java Project

to refine the clustering result. The refined clustering result is passed to the evaluation part, which evaluates the performance of the system based on conventional evaluation techniques employed by Machine Learning Systems.

5.1 Software Tools and Libraries Used

This Java project is programmed in Java SE Development Kit 8. The version of Java is 1.8.0_20. The programming environment is JetBrains IntelliJ Idea Community Edition 2016.1.1. Apache Maven is used to manage the dependencies between the different libraries used for text mining and extraction. The version of Apache Maven is 3.3.9. *Table 5.1* shows the libraries used in the project and their respective roles.

Library/Toolkit	**Role in the Project**
carrot2	Text Mining
jdbc	Database Connection to sqlite database
webMagic	Text Extraction
ini4j	Manage the configuration file
log4j	Manage the output format
java-string-similarity	Compute string similarity
jblas	Linear Algebra Library
Apache math3	Linear Algebra Library

Table 5.1: Libraries and Toolkits used in the Project

The *Carrot 2* Java library [3] is used for the text processing task. Carrot 2 is meant to be an *Open Source Search Results Clustering Engine*. It also provides specialized document clustering techniques. However, it deals only with textual features, and therefore they can't be used in our project. Instead, we had to write our own code for DBSCAN and hierarchical clustering to conform with the structure of our features. The database used is *sqlite* in order to maintain compatibility with Fleming's database, and the *jdbc* library is used to connect and read/write to the database. To create a spider in order to extract full-texts from the E&S database, we use the WebMagic API[4] maintained by Yihua Huang. Java-string-similarity[5] is an implementation of various string similarity and distance algorithms: Levenshtein, Jaro-Winkler, n-Gram, Q-Gram, Jaccard index, longest common subsequence edit distance, cosine similarity maintained by Thibault

[3] http://project.carrot2.org/source-code.html
[4] https://github.com/code4craft/webmagic
[5] https://github.com/tdebatty/java-string-similarity

Figure 5.2: Software Tools and APIs used in InvIdenti

Debatty. *Jblas*[6] library by Mikio Braun and *Apache math3*[7] are popular and widely used linear algebra libraries which we use for matrix operations in latent semantic indexing and logistic regression. Finally, *ini4J*[8] is used to manage the configuration file *invidenti.ini* and *log4j*[9] is used to manage the properties of output messages, eg. color, format etc. which makes debugging easier.

5.2 Phase-wise Project Implementation Description

As mentioned earlier, the project implementation is divided into 6 phases: *Text Extraction, Preprocessing, Logistic Regression, Clustering ,Patent-Publication Matching and Evaluation.* In this section, we will describe the implementation details of each phase along with the sequence diagrams of the project which represent the dynamic behavior of the objects.

5.2.1 Text Extraction

Figure 5.3 shows the class relationship diagram for the text extraction phase. The USPTOSearch class accepts patent number as a parameter and uses this number to create an instance of USPTOSpider class. The USPTOSpider class, in turn, uses the number to request for a patent text to the USPTO patent database by using the PatFt API. If the response of the request is successful, an html file of the patent document is returned. Us-

[6]https://github.com/mikiobraun/jblas
[7]http://commons.apache.org/proper/commons-math
[8]http://ini4j.sourceforge.net/
[9]http://logging.apache.org/log4j/2.x

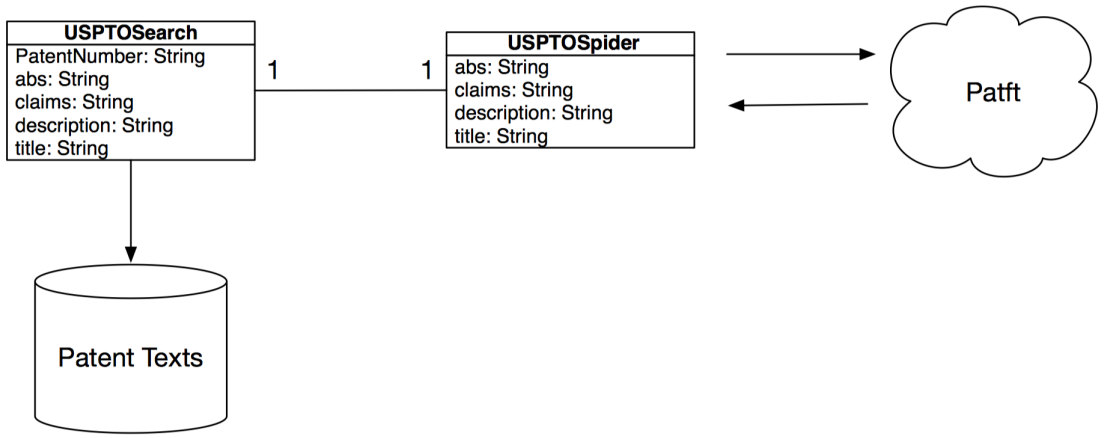

Figure 5.3: Class Relationship : Text Extraction using the PatFt API

ing the WebMagic API, the spider tries to find the title, abstract, claims, and description and returns them to the USPTOSearch class. Sometimes the patent document doesn?t contain the full information (i.e. title, abstract, claims and description), or sometimes the API may not give a successful response due to a network error. In these cases, the spider will make the request thrice, and if all of them fail, the texts will be assigned a null value instead. After extracting the full texts, USPTOSearch class stores the texts in a folder serialized by the patent number. Null values are stored as empty strings.

5.2.2 Preprocessing

In the preprocessing phase, the extracted features from the patent document are converted into document vectors after text processing steps. The class relationship is demonstrated in *Figure 5.4*.

The patent information is extracted from the E&S database, followed by additional information such as location, assignee etc. from Fleming's database. This data, along with the text extracted in the text extraction phase, are aggregated to form the patent instances as shown in the diagram. The main preprocessing steps such as stop words removal, stemming, term frequency calculation and singular value decomposition are performed when the instance of the patent class is passed to *PatentPreprocessingTF* class. The language in the text document needs to be identified as well, and the default language for the patent documents from USPTO is English. The *PatentPreprocessingTF* class then uses the carrot2 and Apache math packages to perform the text processing tasks, and stores the values back into the patent instance as their matrix representation.

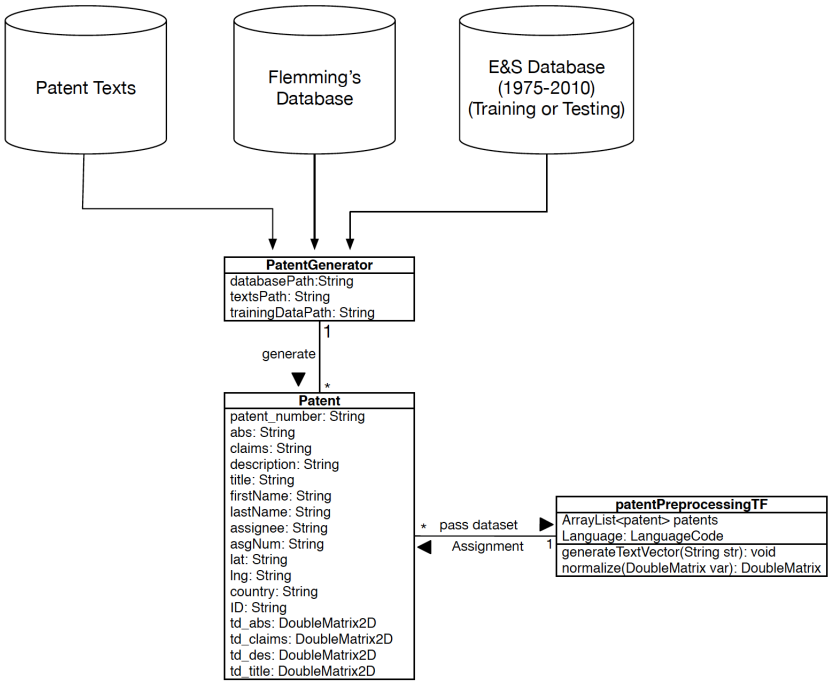

Figure 5.4: Class Relationship : Preprocessing

5.2.3 Logistic Regression

The entire training sequence is illustrated by the Sequence diagram in *Figure 5.6*. The *LRWithBoldDriver* class is the main class that performs the Logistic Regression. It is a child class of *ParameterLearning*, and this layered architecture provides ample scope for additional training mechanisms/techniques to be added to the project in future. *LRWithBoldDriver* essentially implements the Logistic Regression using the Bold Driver Method and Stop Early technique that we described in the previous chapter. The *separateDataset* function is used to divide the patent-instances into training data and validation data for the Stop-Early technique. Once the training is done and the weights of the features are obtained, the distances between patents needs to be generated. This is achieved by the the *CosDistance* class which is a child class of *AbstractDistance*. Again, this layering ensures that the fundamental software engineering aspects such as the *Open Closed Principle* and *Dependency Inversion Principle* are taken care of. The *CosDistance* class makes use of the pCorrelation variable, which is a boolean that determines if the function is actually a distance(true) or a similarity(false) function.

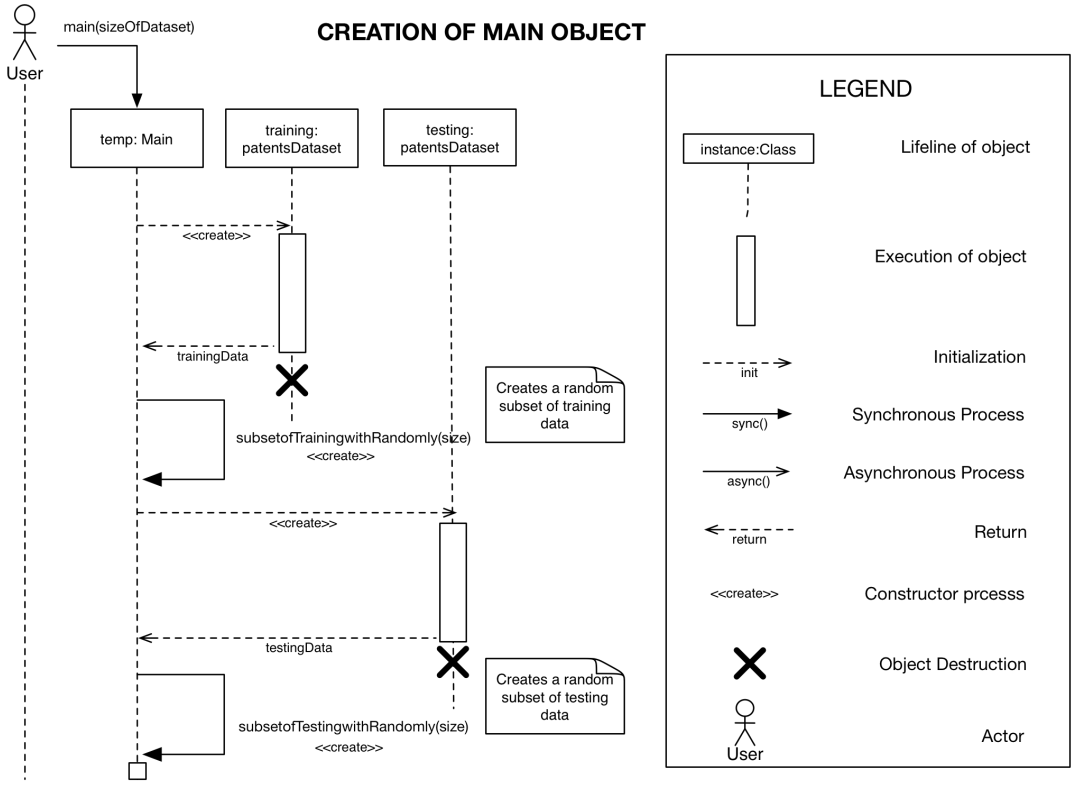

Figure 5.5: Sequence Diagram : Creation of Main Object

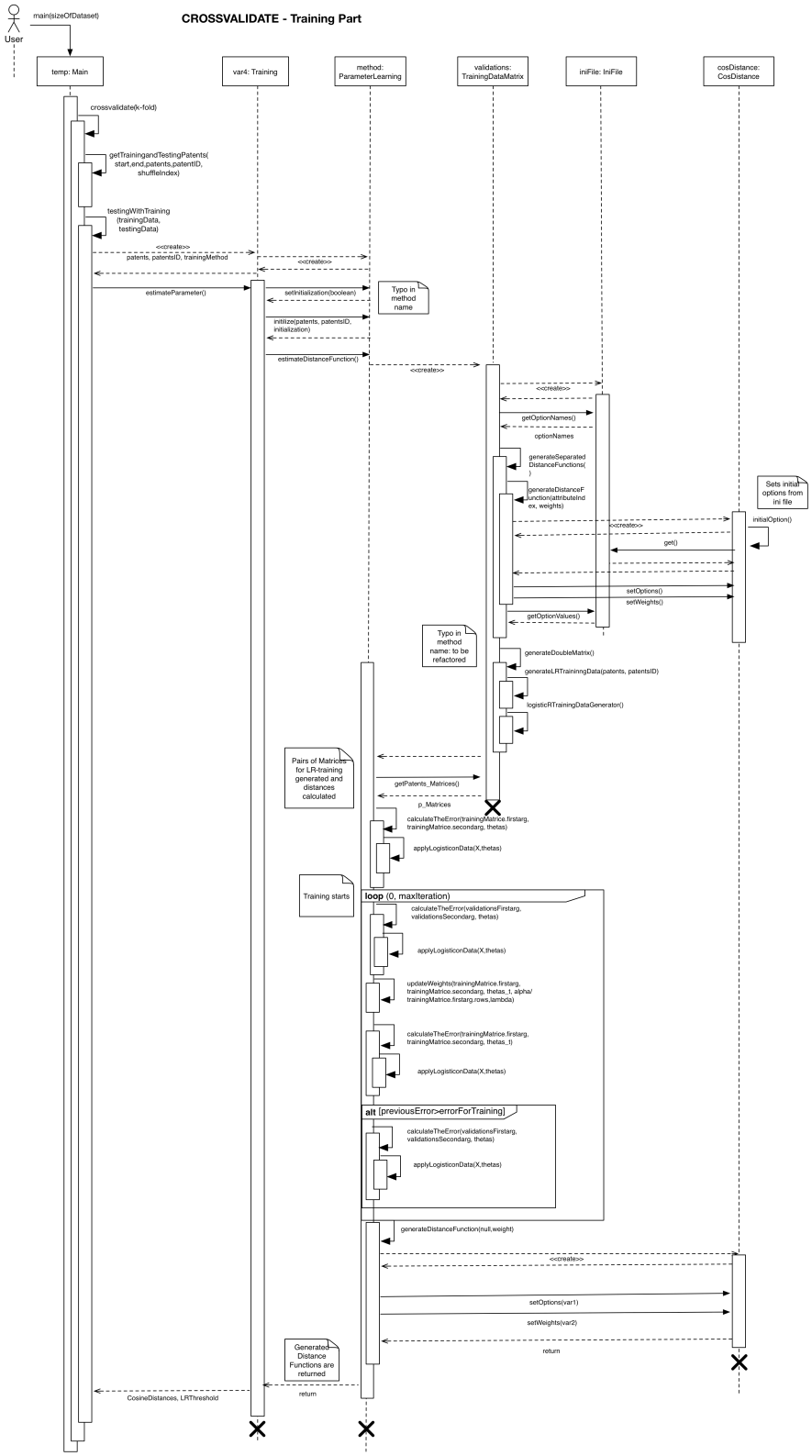

Figure 5.6: Sequence Diagram : Crossvalidate

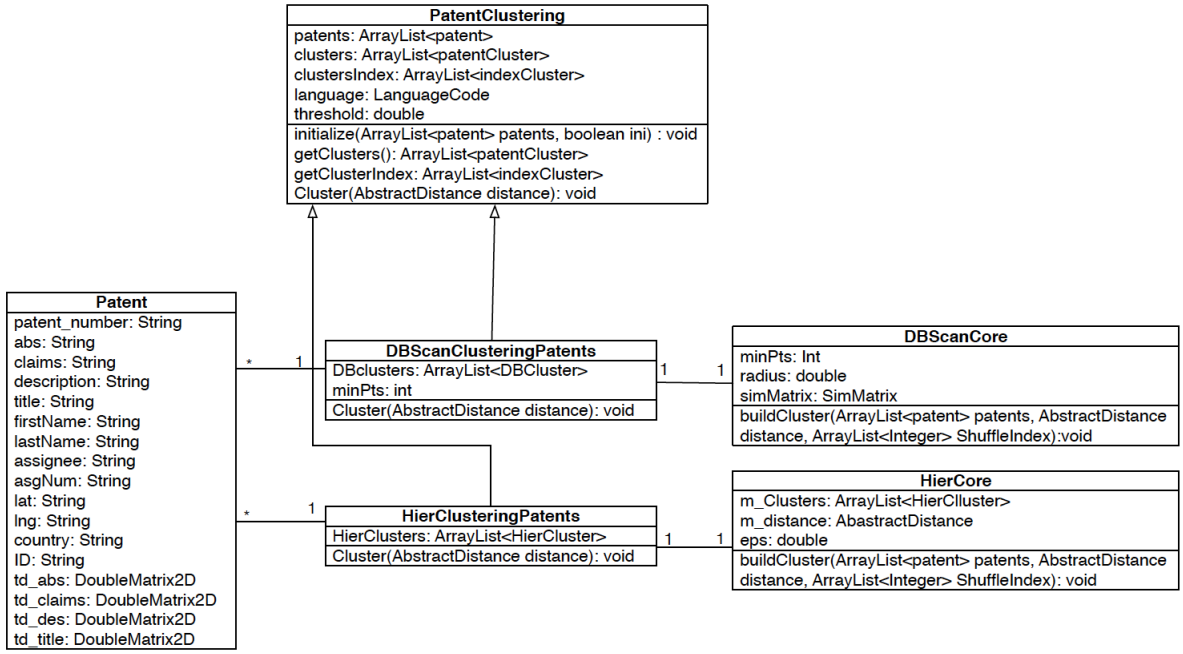

Figure 5.7: Class Relationship : Hierarchical and DBSCAN Clustering

5.2.4 Clustering

The next phase after Logistic Regression is Clustering. Currently, we use two clustering algorithms: Hierarchical Clustering and DBSCAN. As we have discussed earlier, Transitivity is one of our desired side-effects of clustering leading to an enhanced matching of the patents. *Figure 5.7* illustrates the class relationship diagram of the clustering phase. The bulk of the logic for the individual clustering processes is contained in the *DBScanCore* and *HierCore* while the helper classes *DBScanClusteringPatents* and *HierClusteringPatents* contain the parameters and the cluster classes which store the final clusters obtained by clustering. After the clustering is performed, the results are returned and stored into variables in the two helper classes.

5.2.5 Patent-Publication Matching

Patent-Publication matching is a part of the refinement process in *InvIdenti*, where two or more clusters with the same author names are merged based on the similarity of their publication texts. After the clustering is performed, the clusters are stored as instances of the *PatentCluster* class. It uses an ArrayList to list all the patent

instances in the cluster. The author of these corresponding clusters are found using *findTheName* function. Further, we use hashing to find if there's a repetition in the author names, and cases where there is repetition would be candidates for a merge. The author names retrieved by the hashing function are then passed on as keywords from extract information from the *Europe PMC API.*

The API in turn returns an XML file which contains possible matching publication information. A spider designed specially to parse the XML files is deployed, and the publication information is passed individually to be processed by the *processOneResult* function. The function stores additional information about the publication, such as Abstract, Title, Affiliation etc into an XML file. These features are then checked for a match using their individual matching functions, and if a match is found then the clusters are merged.

5.2.6 Evaluation

The final phase of *InvIdenti* is concerned with the quantifying the performance of the software using several metrics. A very commonly used metric for ML systems is the *F-Measurement*, which is the harmonic mean of its precision and recall values. A higher recall value is generally desired. The final clusters obtained after clustering and patent-publication matching are compared again true inventor-identification information. The function *getFscoreofClustering* is entrusted with the job of calculating the false positive/negative and true positive/negative, and in turn, the F-measurement which is the principal metric for our evaluation. Two more metrics that we calculate are called *Lumping Error* and *Splitting Error*. Lumping Error occurs when instances from different patent-inventors are grouped into the same cluster. On the other hand, Splitting error occurs when instances from the same person are grouped into different clusters.

6 Results

In this chapter, we discuss the evaluation of the results of inventor name disambiguation using our approach *InvIdenti*. We will test the performance against standard labelled datasets to ascertain the degree of success of our approach. We have used two datasets for the same, namely the database provided by Fleming et al. [6] and the *Engineers and Scientists (E&S)* database provided by Ge et al. [19]. Further in this chapter, we will introduce the datasets, discuss the evaluation methods and metrics used and analyze the results.

6.1 Datasets

The *E&S* labelled dataset has 96, 104 labeled patent-inventor records with 14,293 unique engineers and scientists with patents. The dataset was provided by Ivan Png for the *PatentsView Inventor Disambiguation Workshop* organized by USPTO. As mentioned earlier, it is derived from the work done by Ge et al. [19]. The dataset provided by Fleming et al. [6] contains comprehensive data about USPTO patents from 1975 to 2010. It provides a lot of additional metadata that can't be directly processed from the patent documents themselves, such as patent assignee number, latitude and longitude of the patent inventor. These metadata are essential features in our approach *InvIdenti*. Taking into account both Fleming's database and E&S database, we have pertinent data for 32, 495 inventor-patent units. For these patents, the full-text is extracted and segmented into title, abstract, claims and description using the patent full-text search engine developed by USPTO. The publication information is extracted from Europe PMC using its API. Since Europe PMC contains only bio-medical publications, the subset of the final dataset which extracts the publication information by using the technology class from USPTO contains 3605 patent-inventor units.

6.2 Measurement Criteria

We use 5 metrics for evaluate the performance of our inventor disambiguation approach *InvIdenti*, namely, *Precision, Recall, F-measure, Lumping Error and Splitting Error*.

The four basic indicators which are used to calculate these metrics are *True Positives, True Negatives, False Positives and False Negatives* as shown in *Table 6.1.*

	Same Cluster	**Different Clusters**
Same ID	True Positive (TP)	False Negative (FN)
Different IDs	False Positive (FP)	True Negative (TN)

Table 6.1: Performance Measurement Indicators

After calculating these indicators on the basis of the standard labelled dataset, we calculate the other metrics using the following formulae.

$$Precision = \frac{TP}{TP + FP} \tag{6.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

$$F\text{ - }Measurement = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{6.3}$$

$$Lumping\ Error = \frac{FP}{TP + FN} \tag{6.4}$$

$$Splitting\ Error = \frac{FN}{TP + FN} \tag{6.5}$$

In this application, *Precision* measures the inventor-patent instances in all the clusters which actually have the same inventor as a fraction of all inventor-patent instances classified into clusters. *Recall*, on the other hand, measures the fraction of invent-patent instances with the same ID which have been put into the same cluster. Generally, neither precision nor recall individually can give a good measure of performance and a compound score called *F-Measurement*. F-measure is the harmonic mean of precision and recall, and takes into account the effects of both the measures. A high F-measure is an indicator of good classification/clustering performance. The other two metrics, namely *Lumping Error* and *Splitting Error* are also related to precision, recall and F-measure. Lumping Error occurs when instances from different inventors are grouped in the same cluster, Splitting error, on the other hand, occurs when instances from the same person are assigned to different clusters. A low splitting as well as lumping error would be desirable in our application.

6.3 Evaluation

The computer used for performance analysis of the software was Apple MacBook Pro Retina (Late 2013) running OSX El Capitan Version 10.11.6 with 2.4 GHz Intel Core i5 Haswell Series processor with 3M cache. The onboard main memory is 8GB 1600 MHz DDR3 RAM and 1536 MB of Intel Iris graphics. The aggregated inventor-patent instances are divided into two datasets, namely *Training* dataset and *Testing* dataset. For the training dataset, 8000 inventor-patent instances are selected at random, and the rest of the instances are used as the testing dataset. First, we train the model using the training dataset and run the cross-validation function in order to evaluate the performance of Logistic Regression. Then we use the values of the parameters obtained using Logistic Regression to test the performance of clustering on the testing dataset. We also compare the performance improvements made to *InvIdenti* to the conventional single-threaded technique used before.

6.3.1 Cross-Validation

In the first part of evaluation, the training dataset is used for cross validation. Each time, subsets of the training dataset with varying sizes are randomly, and a five-fold cross validation is performed on the extracted subset. For each iteration, the logistic regression step is performed first to obtain the weights of the features and the threshold, and these weights and the threshold are used by Hierarchical clustering and DBSCAN. The average of the five-fold cross-validation is then used to measure the performance evaluation metrics. For the varying sizes of the subsets, the mean error values and precision, recall and f-measure are calculated to test the stability of the logistic regression against clustering. The hierarchical clustering uses the *Single-Linkage* clustering technique and the minPts parameter of DBSCAN is initially set to 1 first part of evaluation. The results obtained are shown by the graphs in *Figure 6.1.*

In the graphs, the plotted points are the mean values for the measurements across the different iterations.For Hierarchical clustering, the *Single-Linkage* Method, *Average Group Linkage* Method as well as *Complete Linkage* Method were implemented and evaluated, and the results are plotted in *Figure 6.2.* Performance-wise, the Single-Linkage approach to transitivity yields the best F-measure score, and therefore we use Single Linkage for clustering. On the other hand, for DBSCAN clustering method, the value of *minPts*, which is the minimum number of points required to form a dense region, is a determining factor for the performance. In our analysis, we find that when minPts

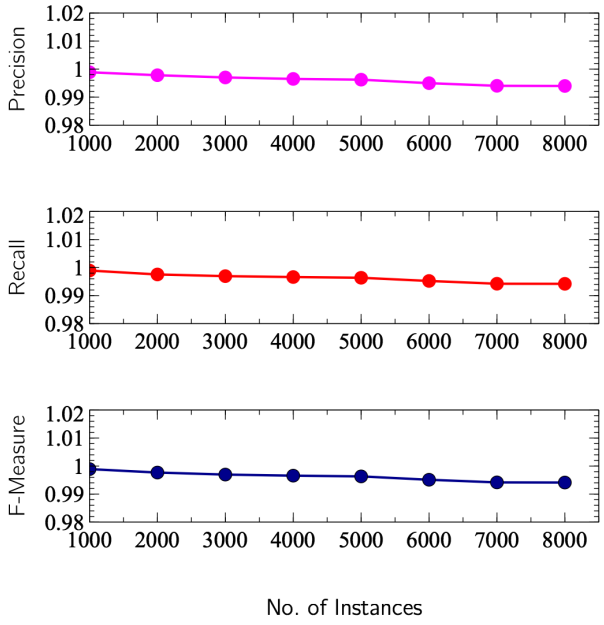

Figure 6.1: Precision, Recall and F-Measure for Crossvalidation

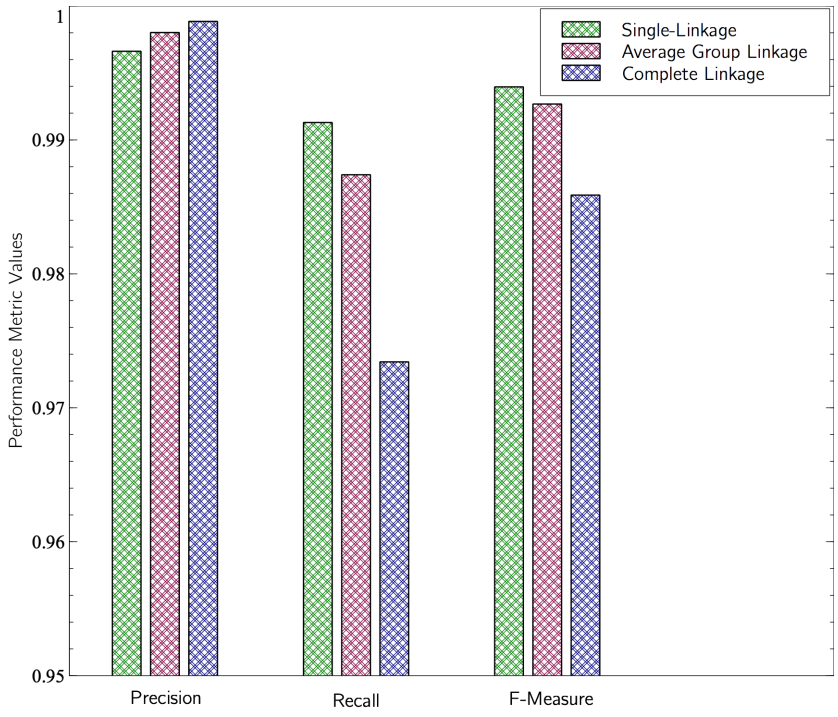

Figure 6.2: Performance Metrics for different Transitivity Options

is set to 1, the number of core objects is the maximum, and so is the F-score. As minPts increases, the number of core objects decreases and the F-score keeps on falling. When minPts becomes 30, the number of core objects becomes 0, i.e., each instance is put into its own cluster

The evaluation shows that for the cross-validation step, the precision, recall and F-measure are always close to 0.99 and the standard error is always less than 0.01. This indicates that there is a very strong inter-relation between the weights and threshold obtained by Logistic Regression and the parameters used by clustering. The values of the parameters obtained for different training dataset sizes shows that some inherent bias in the dataset creeps in for smaller dataset sizes, and the values fluctuate, however, for $\geq$ 6000 patents, the values start becoming very stable. It is also noted that the performance of Hierarchical Clustering and DBSCAN is identical, which can be attributed to the similarity of the way the parameters from Logistic Regression affect the transitivity and the clustering.

6.3.2 Evaluation using Testing Dataset

For the second part of the evaluation, the best-performing approaches of Single-Linkage Clustering and the DBSCAN with minPts are chosen for the evaluation using the entire testing dataset. The values for the weights and threshold are kept the same. Varying sizes of the subsets of the testing dataset are randomly chosen to test the clustering performance by calculating the five performance-analysis metrics. The size of the subset is varied from 2000 to 24495 with an increment of 2000. The values of the precision, recall and F-measure are around 0.99 whereas the lumping error and splitting error is less than 0.02. The subset with size 2000 has the worst F-measurement of 0.988 and the largest splitting error of 0.0159. The final average values of the performance analysis metrics, when evaluated on the entire testing dataset, is tabulated in *Table 6.2*, along with the results of the USPTO PatentsView Inventor Disambiguation Workshop. However, it is worth noting that our tests are run only on a subset of the full E&S dataset, and therefore this table does not reflect an objective comparison between the performance of our algorithm and that used at the USPTO PatentsView Inventor Disambiguation Workshop.

6.3.3 Evaluation on a Benchmark Dataset

Fleming et al. [6], in their research, made use of a benchmark dataset to test the performance of their approach. This dataset contained 95 American inventors and 1332

	Precision	Recall	F-Measure	Lumping Error	Splitting Error
InvIdenti	0.98522	0.99788	0.99151	0.01497	0.00212
USPTO PatentsView	1.0	0.96616	0.98279	-	-

Table 6.2: Performance-Analysis Metrics for *InvIdenti* and *USPTO PatentsView*

associated inventor-patent instances. Their research uses different 'blocking rules' to avoid the generation of the full $\binom{n}{2}$ instances for training of the model. On the basis of these rules, they give two different results for their performance, namely *Lower Bound* and *Upper Bound*. We ran our algorithm on Fleming et al.'s benchmark dataset and tried to compare the results of the their approach versus ours. The results were overwhelmingly positive, as our approach was able to successfully disambiguate all inventor-patent instances and assign them to clusters marked by their respective authors. The results are tabulated in *Table 6.3*.

	F-Measure	Lumping Error	Splitting Error
Fleming et al. (Upper Bound)	0.9744	0.0150	0.0357
Fleming et al. (Lower Bound)	0.9764	0.0150	0.0319
InvIdenti with DBSCAN	1.0	0.0	0.0
InvIdenti with Hierarchical Clustering	1.0	0.0	0.0

Table 6.3: Evaluation results on Benchmark Dataset provided by *Fleming et al.*

6.4 Results of Performance Improvements

In *Section 4.5.1* we have already discussed the merits and necessity of performance improvements in *InvIdenti*. In particular, multi-threading is extremely useful as it can not only aid us in doing I/O and computation in parallel, but also helps to fully utilize the multi-processor architecture of modern day computers. However, since the project was not built to be parallelized from ground-up, and indeed, parallelization was an afterthought on InvIdenti after being confronted with large amount of data processing, the process itself was quite challenging, and involved modifications in the data structures themselves. We were also initially confronted with *NullPointerExceptions* at several junctures of the training process since we were not able to efficiently control thread synchronization and message-passing.

However, once the implementation was successful and we started running performance-evaluation tests, the results were quite encouraging. Not only was the implementation up to 50% faster, but the CPU utilization was also significantly better. In fact, the

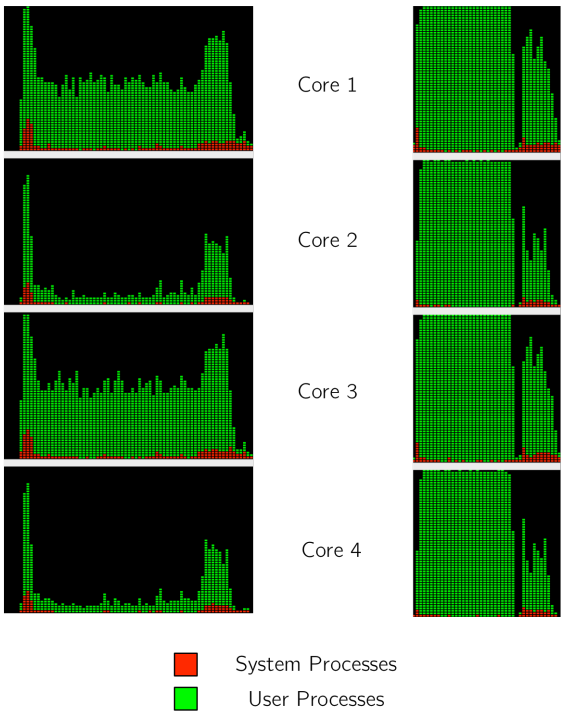

Figure 6.3: Comparison of Single-Threaded and Multi-Threaded SVD

SVD, the most resource intensive portion of our implementation, was able to utilize up to 100% of the available CPU, making it extremely efficient. *Figure 6.3* shows the CPU usage comparison between the single-threaded and the multi-threaded implementation, where the graph on the left is the single-threaded implementation whereas the one on the right is the multi-threaded one. *Figure 6.4* shows the actual times taken by the cross-validate functions with the two different approaches, and the difference is quite stark and significant as the number of instances increases.

For the parallelization of the *SimMatrix* class, the use of worker threads was a seemingly obvious choice, as the class consisted of operations that were atomic but plenty. Worker Threads would also have had a much lesser overhead than if we had to build and destroy threads for each of the atomic processes. Having a thread for each of the atomic processes would not have been a good idea because the performance on multi-threaded CPUs generally increases up to certain number of threads (generally equal to the number of logical cores) and then decreases as the book-keeping overhead starts becoming significant. The final implementation made use of *Thread Pools* and *Cyclic Barriers* and reaped significantly better results than the single-threaded implementation.

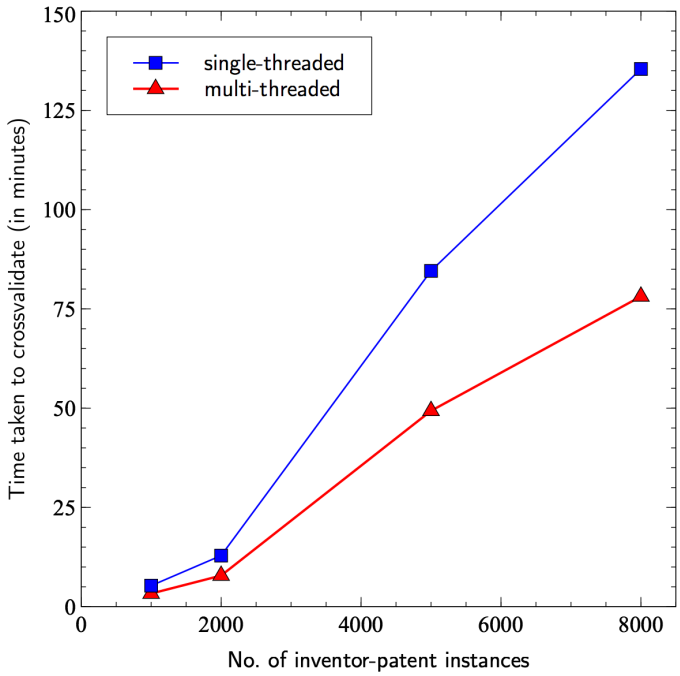

Figure 6.4: Comparison of Running Times on Single-Threaded and Multi-Threaded Implementation

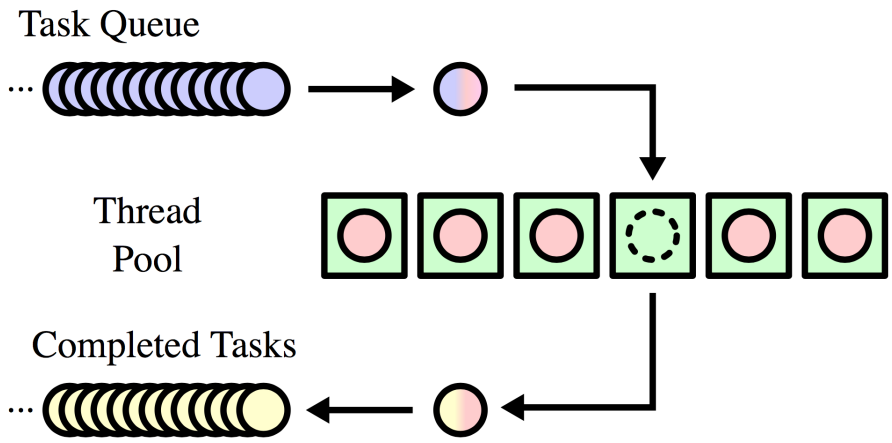

Figure 6.5: Working of Thread Pools

7 Conclusion and Scope for Future Work

In this thesis, we have introduced and described a novel method for inventor disambiguation from patents called as *InvIdenti*. InvIdenti is unique in many ways, most striking being its utilization of a comprehensive set of both metadata and textual features, a transitivity property realized with the help of clustering which gives commendable results in comparison to other approaches by contemporary researchers. The use of Logistic Regression to train the model and get the weights and threshold and the intelligent re-use of these parameters to drive the clustering algorithms makes it much more effective than similar algorithms that use manual tuning. In addition, the software code itself is clean, intelligible and upholds the *S.O.L.I.D* principles of Software Engineering, which makes it extremely convenient to add implementations of better algorithms for training/clustering once they are discovered. Availability of lucid documentation in the form of Javadoc^{TM}documentation and project wiki makes it easier for developers to test, maintain and add functionality to the existing code base. *InvIdenti* also makes use of several optimization techniques just as a multi-threaded Latent Semantic Indexing, Bold Driver technique to automatically adjust the learning rates and Stop-Early technique to avoid overfitting during training. An added layer of Patent-Publication Matching doesn't appear to increase accuracy of the clustering in our experiments, however this is because of the fact that the clustering algorithms themselves give very optimal results. Nevertheless, Patent-Publication Matching is expected to be more effective when the number of authors having similar names increase.

The scope for future work on InvIdenti is immense. First and foremost, the project is not designed to run on a distributed cluster computing network such as *Apache Spark*. As a result, on the computers that we tested the code on, we were able to train the model for a maximum of 10,000 patents. This is also because of the fact that the training process generates all pairs of inventor-patent instances exhaustively, giving rise to $\frac{n(n-1)}{2}$ combinations for n inventor-patent instances. This gives rise to a huge number of instances, which makes processing very resource-intensive on conventional single-CPU computers. Although techniques such as blocking, mini-batch, stochastic gradient descent have been proposed to accelerate the training process, they come with major side-effects, most notably a decline in accuracy of the model. Therefore, extending this

project on *Big Data* as well as *Distributed Platforms* would enable us to train and test the model for the full dataset and the ability to handle future updated patent databases as well. Furthermore, our implementation makes use of a parallelized Similarity Matrix generation, however the clustering algorithms themselves are largely sequential in nature. By making use of the message-passing interface, the clustering algorithms can be made multi-threaded, which would lead to a much better performance. Also, our implementation was seen to yield best results for *single-linkage clustering* in Hierarchical clustering and *minPts* set to 1 in DBSCAN, both of which aim at decreasing splitting error. However, if the number of inventors in the patent database becomes massive, this approach could lead to a rise in lumping error and therefore, the clustering parameters should be re-evaluated for such a database. Last but not least, the true potential of this project can be unleashed if the inventor-clusters can ultimately be associated with some semantic knowledge in order to aggregate the best possible pool of experts for a complex medical task, as mandated by the project Mi-Mappa [1]

Bibliography

[1] RWTH Aachen Lehrstühl für Informatik 5. mi-mappa. `http://dbis.rwth-aachen.de/cms/projects/mi-mappa`. Accessed: 2016-03-15.

[2] Eric Sven Ristad and Peter N Yianilos. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532, 1998.

[3] Anna Huang. Similarity measures for text document clustering. In *Proceedings of the sixth New Zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[4] Lei Sun. Clustering of patents for inventor identification. Master's thesis, Lehrstühl für Informatik 5, RWTH Aachen University, March 2016.

[5] Robert C Martin. *Clean code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2009.

[6] Guan-Cheng Li, Ronald Lai, Alexander D'Amour, David M Doolin, Ye Sun, Vetle I Torvik, Z Yu Amy, and Lee Fleming. Disambiguation and co-authorship networks of the us patent inventor database (1975–2010). *Research Policy*, 43(6):941–955, 2014.

[7] Object Management Group. The unified modeling language. `http://www.uml.org`. Accessed: 2016-05-13.

[8] Object Management Group. What is uml? `http://www.uml.org/what-is-uml.htm`. Accessed: 2016-05-13.

[9] Oracle Inc. Javadoc - the java api documentation. `http://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html`. Accessed: 2016-03-18.

[10] Stéphane Maraut and Catalina Martínez. Identifying author–inventors from spain: methods and a first insight into results. *Scientometrics*, 101(1):445–476, 2014.

[11] Kevin W Boyack and Richard Klavans. Measuring science–technology interaction using rare inventor–author names. *Journal of Informetrics*, 2(3):173–182, 2008.

[12] Bruno Cassiman, Patrick Glenisson, and Bart Van Looy. Measuring industry-science links through inventor-author relations: A profiling methodology. *Scientometrics*, 70(2):379–391, 2007.

[13] Neil R Smalheiser and Vetle I Torvik. Author name disambiguation. *Annual review of Information Science and Technology*, 43(1):1–43, 2009.

[14] Oracle Inc. Javadoc - thread objects. `https://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html`. Accessed: 2016-03-23.

[15] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[16] Caspar J Fall, Atilla Törcsvári, Karim Benzineb, and Gabor Karetka. Automated categorization in the international patent classification. In *ACM SIGIR Forum*, volume 37, pages 10–25. ACM, 2003.

[17] Oracle Inc. Concurrency. `https://docs.oracle.com/javase/tutorial/essential/concurrency/`. Accessed: 2016-06-07.

[18] Roberto Battiti. Accelerated backpropagation learning: Two optimization methods. *Complex systems*, 3(4):331–342, 1989.

[19] Chunmian Ge, Ke-Wei Huang, and Ivan PL Png. Engineer/scientist careers: Patents, online profiles, and misclassification bias. *Strategic Management Journal*, 37(1):232–253, 2016.