Instance-based ontology alignment using decision trees

by

Tahereh Boujari

LIU-IDA/LITH-EX-A--12/055—SE

2012-10-17
Final Thesis

Instance-based ontology alignment using decision trees

by

Tahereh Boujari

LIU-IDA/LITH-EX-A--12/055—SE

2012-10-17

Supervisor:  Valentina Ivanova
Examiner: Patrick Lambrix
Abstract

Using ontologies is a key technology in the semantic web. The semantic web helps people to store their data on the web, build vocabularies, and has written rules for handling these data and also helps the search engines to distinguish between the information they want to access in web easier. In order to use multiple ontologies created by different experts we need matchers to find the similar concepts in them to use it to merge these ontologies.

Text based searches use the string similarity functions to find the equivalent concepts inside ontologies using their names. This is the method that is used in lexical matchers. But a global standard for naming the concepts in different research area does not exist or has not been used. The same name may refer to different concepts while different names may describe the same concept.

To solve this problem we can use another approach for calculating the similarity value between concepts which is used in structural and constraint-based matchers. It uses relations between concepts, synonyms and other information that are stored in the ontologies. Another category for matchers is instance-based that uses additional information like documents related to the concepts of ontologies, the corpus, to calculate the similarity value for the concepts.

Decision trees in the area of data mining are used for different kind of classification for different purposes. Using decision trees in an instance-based matcher is the main concept of this thesis. The results of this implemented matcher using the C4.5 algorithm are discussed. The matcher is also compared to other matchers. It also is used for combination with other matchers to get a better result.
Acknowledgement

Many Thanks goes to
  Patrick Lambrinx, for great advices and kind helps.
  Valentina Ivanova, for her supervising.

Also,
  Javeriya Riaz, for the opposition.
  My parents: for their endless love.
  Family and friends, for their kind words to give me courage to continue my work.
# Contents

1 Introduction 3  
  1.1 Motivation ............................................. 3  
  1.2 Outline .................................................. 4  

2 Ontology alignment 5  
  2.1 Ontology definition ..................................... 5  
  2.2 Ontology alignment ...................................... 7  
  2.3 Document-based ontology alignment ..................... 9  

3 PubMed 10  
  3.1 Definition ............................................... 10  
  3.2 Searching strategies ................................... 10  
  3.3 Automated search ....................................... 12  

4 Decision Trees in Weka 13  
  4.1 Decision trees .......................................... 13  
     4.1.1 Definition ............................................ 13  
     4.1.2 Construction ......................................... 14  
     4.1.3 Decision tree classifier ............................ 15  
  4.2 Feature vectors ......................................... 17  
     4.2.1 Supervised classification ........................... 17  
     4.2.2 Classification features .............................. 17  
  4.3 Weka ...................................................... 18  
     4.3.1 ARFF Files ............................................ 18  
     4.3.2 Classifier output .................................... 19  

5 Decision Tree matcher 22  
  5.1 Method .................................................. 22  
  5.2 Implementation steps ................................... 23  
  5.3 Implementation .......................................... 26  
  5.4 Limitations .............................................. 26
6 Experiment 28
6.1 Basic concepts ........................................... 28
  6.1.1 Dataset ............................................. 28
  6.1.2 Measures definition ................................ 28
  6.1.3 Classifier attributes ............................... 29
  6.1.4 Document based options ........................... 30
  6.1.5 Combination with other classifiers ............... 30
6.2 Analysis .................................................. 31
  6.2.1 Test result for DT classification-whole corpus .. 31
  6.2.2 Test result for DT classification- part of corpus: 32
  6.2.3 Comparison with other classifiers ................. 33
  6.2.4 Combination with other classifiers ............... 34
  6.2.5 Parent concepts and minimum number of instances per leaf ........................................ 35
  6.2.6 Confidence factor ................................... 36
  6.2.7 Threshold and corpus size ........................ 37
6.3 Conclusions ............................................... 38

7 Discussions 39
7.1 Conclusion ............................................... 39
7.2 Further work ............................................ 39

A Implemented Functions Description 43

B Setup Guide for the System 46
  B.1 Setup SAMBO-DeskApp Project in NetBeans IDE .... 46
  B.2 Run SAMBO-DeskApp application (without IDE) ... 47
  B.3 Run the matchers ....................................... 47
  B.4 To inspect the code .................................. 48

C Nomenclature 49
List of Tables

3.1 Advanced search with field tags ........................................ 11
4.1 Part of data table for MA_0001113 class "meninges" concept in Mouse Anatomy ........................................ 18
5.1 Suggestion matrix ............................................................. 24
6.1 Scheme-specific options for the J4.8 decision tree learner ..... 29
6.2 Result of classification with whole corpus ......................... 31
6.3 Result of classification with part of corpus ......................... 33
6.4 Precision for different Parent concepts and min number of instances per leaf ........................................ 36
6.5 Compare precision for different confidence factor ................. 37
6.6 Effect of threshold on different size of corpus ..................... 37
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Visualization of three concepts in the adult mouse anatomy</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Two concepts, or classes as they are called in OWL</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Example of ontology overlapping</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Alignment strategy</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>PubMed search engine</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Basic search query</td>
<td>12</td>
</tr>
<tr>
<td>3.3</td>
<td>Basic download query</td>
<td>12</td>
</tr>
<tr>
<td>4.1</td>
<td>Decision tree for the weather data</td>
<td>14</td>
</tr>
<tr>
<td>4.2</td>
<td>The Pseudo-code for building a decision tree</td>
<td>14</td>
</tr>
<tr>
<td>4.3</td>
<td>Example of subtree raising</td>
<td>16</td>
</tr>
<tr>
<td>4.4</td>
<td>Part of arff file for MA_0001113 &quot;meninges&quot; concept in Mouse Anatomy</td>
<td>19</td>
</tr>
<tr>
<td>4.5</td>
<td>Visualize tree for J48 classifier example</td>
<td>20</td>
</tr>
<tr>
<td>4.6</td>
<td>J48 classifier output for part of 4-classes Mouse Anatomy</td>
<td>21</td>
</tr>
<tr>
<td>5.1</td>
<td>Algorithm 1- Build Corpus from PubMed PubMedCorpus-Builder.buildCorpus</td>
<td>23</td>
</tr>
<tr>
<td>5.2</td>
<td>Algorithm 2-Create feature vectors from corpus DecisionTree-BuildDatabase.generateCorpusDB</td>
<td>24</td>
</tr>
<tr>
<td>5.3</td>
<td>Algorithm 3-Build classifier</td>
<td>25</td>
</tr>
<tr>
<td>5.4</td>
<td>Algorithm 4-Evaluate the result with different threshold evaluateStrategies</td>
<td>26</td>
</tr>
<tr>
<td>6.1</td>
<td>Combination and filtering</td>
<td>31</td>
</tr>
<tr>
<td>6.2</td>
<td>Comparison of classification with whole corpus and part of corpus</td>
<td>32</td>
</tr>
<tr>
<td>6.3</td>
<td>Comparison between C4.5 and other matchers with threshold (= 0.8)</td>
<td>34</td>
</tr>
<tr>
<td>6.4</td>
<td>C4.5 part of corpus compared to other matchers with &amp; without combination</td>
<td>35</td>
</tr>
<tr>
<td>6.5</td>
<td>C4.5 whole corpus compared to other matchers with &amp; without combination</td>
<td>35</td>
</tr>
</tbody>
</table>
6.6 Using parents’ instances with different min number of instances per leaf .......................... 36
6.7 Confidence factor effect on precision ................................. 37
6.8 Precision for different threshold and corpus size .................... 38
6.9 Recall for different threshold and corpus size ....................... 38
Chapter 1

Introduction

1.1 Motivation

Nowadays using a simple way of search which mostly is text-based search of classic "web of documents" will not cover the need of users. For the search engines the text in the web pages without any specific tag or label is just a series of characters. But if we store the important part of the information in a different way the engine could distinguish between the necessary parts and access the information easier and smarter. This new approach, "web of data", which is changing and storing the unstructured and semi-structured information in web in a structured way is called semantic web. According to international standards organization for the WWW [20], the definition of Semantic Web is the linked data over network that can be used by machines to extract these data. It helps people to store their data on the web, build vocabularies, and have written rules for handling these data.

Ontologies are one of the main tools in the semantic web which contain different concepts in one specific area with their attributes. The number of resources that are stored in this format is growing so fast; Swoogle\footnote{http://swoogle.umbc.edu/} has indexed over 10,000 of them. In order to use multiple ontologies created by different experts we may need a method to find the similar concepts and after mapping or aligning these concepts, merge these ontologies by using the alignment to have one reference for both ontologies.

Text based searches use the string similarity functions to find the equivalent concepts using their names. But a global standard for naming the concepts in different research area does not exist or has not been used. The same name may refer to different concepts while different names may describe the same concept [3]. The other approach for calculating the similarity value between concepts is by using the other information stored in ontology like the names of the predecessor and successor concept which creates the structure of ontology, synonyms and etc. Instance based classification use
additional information like documents related to the concepts of ontologies. General approach in this method is that the more documents they have in common the more similar they are.

There are different methods for using these instances and calculate the similarity value. According to [15] and [5] the Ontology mapping techniques are divided into 6 categories:

1. Lexical (similarities between textual descriptions of concepts)
2. Structural (using the structure created by is-a or part-of relations of the ontologies)
3. Constraint-based (using axioms for example suggest two concepts with same range and domain)
4. Instance-based mapping (using instance data for classification)
5. Use of auxiliary information (Dictionaries and thesauri)
6. Combining different approaches

Recently there has been a growing interest for category 4 among others categories [25]. In this paper we implement an instance based matcher that uses decision tree classifier inside the SAMBO system which is a system for aligning and merging biomedical ontologies. We test and compare it with other matchers that are used in this system like Naïve based which is another instance based matcher [5].

1.2 Outline

In this thesis first we discuss the ontologies’ definition and how we could align them and also what are the steps if we use the instance based ontology alignment in chapter 2. For building an instance based classifier we need a corpus, the way we create it will be discussed in chapter 3.

Then we discuss the decision trees and how to create them. What a decision tree classification algorithm needs to create a classifier by introducing the format of input, feature vectors and different settings it could have in Weka suite in chapter 4.

In chapter 5 we use the background knowledge from previous chapters and explain the steps of our implementation of instance based decision tree matcher for SAMBO system. We also will discuss the limitations of our algorithm.

In chapter 6 which is our experiment result we introduce the inputs, used evaluation measures, the attributes of matcher and then in the analysis part we see the effect of different settings on the result. And also we combine our experiment with other matchers inside the SAMBO and analyze the result.

The conclusion of results and further work is discussed in chapter 7.
Chapter 2

Ontology alignment

2.1 Ontology definition

"An ontology is a formal, explicit specification of a shared conceptualization." [1]. In other words an ontology is a domain of knowledge that can contains classes or concepts, their attributes, axioms and relationships between these classes like is-a or part-of relations. Sometimes it contains instances for its classes too. Ontologies are like an index to the repository of information inside that domain. [5]

Ontologies can contain different kinds of information like it was mentioned before; like the relationships between classes. As an example, in Figure 2.1 part-of relations is shown between the three concepts. Ear Bone is part of Tympanic Cavity where Tympanic Cavity itself is a part of Auditory System. Each concept could have more than one sub-concept and this hierarchy could help to find the similar concepts in two different ontologies when they have same relations like part-of or is-a with the equivalent concept (used in structure-based matchers).

Figure 2.1: Visualization of three concepts in the adult mouse anatomy [7]

Lambrix and Tan [5] categorize the different kinds of ontologies as controlled vocabularies, taxonomies, thesauri, data models, frame-based ontologies and knowledge-based ontologies. For instance, we can use W3C Web Ontology Language (OWL) which is "a Semantic Web language designed to
2.1. ONTOLOGY DEFINITION

Chapter 2. ONTOLOGY ALIGNMENT

represent rich and complex knowledge about things, groups of things, and relations between things. OWL is a computational logic-based language such that knowledge expressed in OWL can be reasoned with by computer programs” [21]

Figure 2.2: Two concepts, or classes as they are called in OWL

We can see part of the structure of OWL format in Figure 2.2 which shows two concepts in Mouse ontology [13]. It contains information like names, data type, relations, synonyms and etc.

Some information extracted from Figure 2.2 is:

- Class names: middle ear (ID=MA_0000253), auditory bone (ID=MA_0000254)
- Data type: string
- Relations: auditory bone is part of middle ear (in class MA_0000254 we have UNDEFINED_part_of with value= MA_0000253)
- Synonyms: genid1762 and genid1763 are synonyms of auditory bone.

When we want to merge and reuse this information with other ontologies we may suffer from the problem of overlapped concepts. This means first we should define the equivalence links between the similar terms in ontologies.
then we are able to have an alignment to map and merge these terms and use these ontologies at the same time.

Figure 2.3 shows two small pieces of two ontologies. The equivalent concepts are connected to each other. So if we want to create a single reference for both of them we can merge these concepts and refer to them as a single concept. If we have more than two ontologies it can be repeated several times to get one reference for multiple ontologies at the end.

![Figure 2.3: Example of ontology overlapping](image)

### 2.2 Ontology alignment

The SAMBO system uses ontologies in OWL format as an input and returns the alignment as output. Figure 2.4 shows a general alignment strategy based on the computation of similarity values between concepts in the given ontologies. As shown in the framework the algorithm can use several matchers. They could compute the similarity value based on linguistic functions, structure-based strategies, constraint-based methods, and instance-based strategies.

Auxiliary information like general dictionaries, instance corpora, domain thesauri can be used by them. The result could be the combination of these values with different weights. Filters by pre-defined thresholds can be applied on the suggested list [5]. Then an expert could see the result and accept or reject them. A conflict checker is also used in this step to avoid conflicts that may be occurring by the alignment relationships. This approved list could be used in the alignment algorithm for the rest of the pairs in ontologies because it may influence the further suggestions. The final list will be the output of the system.

The text-based matchers in the current version of SAMBO are NGram, TermBasic and TermWN, the domain knowledge based matcher is UML-SKSearch and the instance-based matcher is BayesLearning. The matching algorithms of text-based matchers are based on the textual descriptions (names and synonyms) of concepts and relations. It uses string matching algorithms like n-gram which is used in NGram matcher. An n-gram is a set of n consecutive characters from a string. When two strings have more n-grams in common it makes them more similar to each other. Another string
matching algorithm is Edit distance which is defined as the number of deletions, insertions or substitutions needed to transform a string to another string and the higher number of actions needed to transform two strings into each other means they are more different. There is also a linguistic algorithm that computes the similarity of the terms by comparing the lists of words of which the terms are composed. The terms are similar when there are a large number of common words in their lists. If we use these three string matching algorithms we have TermBasic matcher. TermWN uses string matching algorithms plus WordNet \(^1\) to find synonyms and is-a relations.

For the other category, domain knowledge based, the matcher UMLSKSearch uses the Metathesaurus in the Unified Medical Language System \(^2\). The similarity of two terms in the source ontologies is determined by their relationship in UMLS. The UMLS Knowledge Source Servers used to query the UMLS Metathesaurus with ontology terms. It assigns a similarity value of 1 for exact matches of query results for the two terms, 0.6 if the source ontology terms are synonyms of the same concept and 0 otherwise. [5]

The instance-based matcher BayesLearning uses the life science literature that is related to the concepts of ontologies as instances, so it may also

---

\(^1\)http://wordnet.princeton.edu/
\(^2\)http://www.nlm.nih.gov/research/umls/
be called document-based matcher. More details about this category are discussed in next section.

2.3 Document-based ontology alignment

The basic steps of a document based ontology alignment in SAMBO are as follows [6]:

1. Generate corpora: For each ontology that we want to align we generate a corpus of documents.

2. Generating classifiers: For each ontology one or more document classifiers are generated. The corpus of documents associated to an ontology is used for generating its related classifiers.

3. Classification: Documents of one ontology are classified by the document classifiers of the other ontology and vice versa.

4. Calculate similarities: A similarity measure between concepts in the different ontologies is computed based on the results of the classification.

Different kinds of instances can be downloaded from different sources. We can use documents downloaded from searching the terms on internet or other databases which are more categorized and indexed based on the subject of information they contain. In SAMBO literature from a search engine for life science library named PubMed is used. The details are discussed in next chapter.
Chapter 3

PubMed

In our implementation we generated a corpus of PubMed documents for different classes using the programming utilities [17] provided by the retrieval system Entrez [9]. This will be used as training and testing datasets. In next part we discuss the details about the search engine, its methods for search and download and how we can automate this process.

3.1 Definition

PubMed [11], is the U.S. National Library of Medicine’s (NLM) premiere search system for health information. MEDLINE [10] as PubMed content is NLM’s database of over 19 million citations of articles published in biomedical and related journals which have been fully indexed [12]. Different searching methods can be used for it. It could be a simple term or with the advance tags to filter the data in the desired way.

3.2 Searching strategies

To search a name in PubMed you could simply type a word or phrase into the query box, this could be the name, subject, author or journal (Figure 3.1). We just search the name of classes from the ontologies. The terms we are searching could be combined with connector words: ”AND”, ”OR” or ”NOT” using upper case letters.

When a document is added to PubMed in addition to information that the user enters like title, author and . . . , the subject analysts examine the article and assign the most specific MeSH terms to it as Major Topics, typically ten to twelve per document.

PubMed could query the database using this information in two ways:

- All fields: PubMed looks first for the words as a MeSH term, then for journal titles, then authors and then rest of the fields.
3.2. SEARCHING STRATEGIES

Figure 3.1: PubMed search engine

- With field tags: The Advanced Searching methods with Field Tags (Table 3.1) can be used for querying the terms [12]. For instance we use just major topics that are assigned to each document in our search by having a specific field tag, the term followed by [MAJR] (e.g. "Neoplasms"[MAJR]). Then it just returns the citations that have that term in their major topics.

<table>
<thead>
<tr>
<th>Category</th>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MeSH Heading</td>
<td>[mh] or [MeSH]</td>
<td>May be added to a term to restrict a search to only MeSH terms (e.g. knee [mh])</td>
</tr>
<tr>
<td>Subheadings</td>
<td>Two letters</td>
<td>Therapy is th, so cancer/therapy and cancer/th are equivalent searches. Use only one subheading per term; combine additional subheadings by using OR (neoplasms/th OR neoplasms/an). A search may also &quot;free float&quot; headings [mh] and subheadings [sh] (e.g., hypertension [mh] AND toxicity [sh])</td>
</tr>
<tr>
<td>Do Not Explode</td>
<td>[mh:noexp] or [sh:noexp]</td>
<td>Use to restrict searches to articles focusing on the broadest MeSH term</td>
</tr>
<tr>
<td>Major Topics</td>
<td>[MAJR]</td>
<td>Use to restrict a search to citations where the term is the major topic or focus of the article. This may be used with Do Not Explode: [MAJR:noexp]</td>
</tr>
</tbody>
</table>

Table 3.1: Advanced search with field tags

Using "All fields" for query a term in the database return more documents because it searches all the fields which make the chance of finding a document related to that term higher. It also may return wrong documents in the way that the term we search is in one of the fields stored in the database but the document is not exactly about that term. On the other hand by using the field tags like major topics, PubMed returns fewer documents by removing most of the wrong suggestions. The reason is it only searches the assigned major topics to the documents.

In other words we can say that the corpus is all the documents that are the result of searching the term in all fields and when we add the tags we are just using part of this corpus which is more related to that term.
3.3 Automated search

The Programming Utilities (E-utils) are a set of eight server-side programs with one interface into the Entrez [9] query and database system at the National Center for Biotechnology Information (NCBI).

The E-utils includes 38 databases covering a variety of biomedical data, including nucleotide and protein sequences, gene records, three-dimensional molecular structures, and the biomedical literature. [16]

For using the E-utils the query (Figure 3.2-Basic Search query) should be sent as a fixed URL syntax. A standard set of input parameters can be set to the necessary values like the database name or the term. The translated query will be sent to NCBI software components to search and retrieve the requested data and return it as list of UID of the documents which match the query.

![Figure 3.2: Basic search query](image)

Then the UID list will be sent as an input to another query (Figure 3.3-Basic download query) which will return the information in retrieval format which was set in this query.

![Figure 3.3: Basic download query](image)

In our case the input database is PubMed, term is the concept name from ontology, retstart is equal to one and retmax is set different numbers and is the size of corpus in test result table in chapter 5. Retmode is set to xml which the abstract text with another class will be save as text files and id list is the list of PubMed ids which are the output of search query.

Each document that may be downloaded with one of searching methods that mentioned before, will be saved in database after transformed into feature vector from function computeFeatureVector(String fpath) in Class Document, after removal of stop words, interpunction and suffix, primitive corpora [6].
Chapter 4

Decision Trees in Weka

In this chapter we discuss the definition of decision tree, how to create it and the algorithms that are used in Weka suite to create it for the purpose of classification different data and the settings that can be set for the classifier to make it ready for the real life problems.

4.1 Decision trees

4.1.1 Definition

Decision trees can be used in a lot of areas and depending on the usage could have different definitions. The definition that can be used in data mining area is: ”A decision tree describes a tree structure where the leaves represent classifications and branches represent conjunctions of features that lead to those classifications ” [8].

For creating the tree we use the training set which is data instances with different attributes as input. Every instance in training set is assigned to a class or category. The purpose of crating the tree is to predict the class of instances in another set which we call testing set. The testing set is data instances with the same attributes of training set with unknown class.

As an example for decision trees, Figure 4.1-Decision tree for the weather data [28] shows a decision tree made from some weather data about decision was made regarding to the value of attributes of training set; outlook, humidity, windy, to whether play or not (yes or no as a two-value class). The main idea is that these attributes create this tree and new instance with the most common attributes will trace it and will end up in the leaf of the most similar class (play) which will be our decision.

So with a new set of data like (outlook=sunny, humidity=normal, windy=ture) according to this tree we predict play=yes. In some cases the decision tree cannot help us to make a decision (outlook=sunny, humidity=low, windy=false), it means the new set of data is not fitting to the tree or
4.1. DECISION TREES

Figure 4.1: Decision tree for the weather data [28]

The methods to eliminate this to happen will be discussed later.

4.1.2 Construction

The function for creating a tree is a recursive function. First we select an attribute as a root for tree from the given training set instances’ attributes with different values as our input. This will divide the training set into several subsets, for each value that root-attribute could have, we have one subset.

Now each subset could be seen as a new training set. Then all of them will have a root attribute and subsets. This will be repeated until the values of attributes of instances do not have the potential to make a new branch. In other words all instances at a node will have the same classification so no new branch is needed. Figure 4.2-The Pseudo-code for building a decision tree shows the main steps of this function.

Figure 4.2: The Pseudo-code for building a decision tree [4]

- Check for base cases
- For each attribute a
  - Find the feature that best divides the training data such as information gain from splitting on a
- Let a_best be the attribute with the highest normalized information gain
  - Create a decision node that splits on a_best

Repeat on the sub-lists obtained by splitting on a_best and add those nodes as children of node.

But how do we decide which attribute should be the root? In [28] a
measure called the information is introduced which is measured in units called bits. As we said before we stop to add a new branch when all the instances could be classified in the last node but it is an ideal situation. For example the training set could have two instances with exact same attributes but different classes. So we may change that condition to stop when the data cannot be split any further.

The information measure is related to the amount of information that we can gain from the decisions that could be made over that attribute when it is the root in the decision tree. These decisions could be made in one or more stages.

Information gain is based on the concept of entropy used in information theory. The function used here as the entropy is:

\[ \text{entropy}(p_1, p_2, \ldots, p_n) = -p_1 \log p_1 - p_2 \log p_2 \ldots - p_n \log p_n \]

where \( p_1, p_2, \ldots, p_n \) are fractions so the minus sign before the logarithm makes the result positive.

This divide-and-conquer approach for creating decision trees which is also known as ID3, was developed and refined over many years by J. Ross Quinlan of the University Of Sydney, Australia [14]. Quinlan mentioned the ID3 algorithm is a robust and practical algorithm.

A series of improvements has been made to ID3 to create a new and open source version for this algorithm which is called C4.5 (the commercial version, C5.0 with more improvement is also available). These improvements include methods for dealing with numeric attributes, missing values, noisy data, and generating rules from trees [28]. Weka suite uses this algorithm for making decision tree classifier by the name of J48 class [24].

### 4.1.3 Decision tree classifier

The data we used has nominal attributes without missing values. But when the algorithm creates the classifier some changes should be applied to make it ready for use on real-world problems.

One of the most important changes is how to prune decision trees. Although trees constructed by the algorithm perform well on the training set, the over-fitting problem still happens for unseen data from testing set when we try to classify them. Pruning reduces the size of the tree and makes the complexity of it lower by removing the sections of it that may be based on noisy or erroneous training set’s data.

**Pruning methods**

There are two kinds of pruning methods [28]: post-pruning which is when we build the complete tree and prune it afterwards and pre-pruning which means trying to decide during the tree-building process when to stop developing sub-trees, which seems more efficient because it would avoid all the
work of developing sub trees only to throw them away afterwards like the way we do in post-pruning.

But in C4.5 post-pruning is used because of some advantages. For instance in some situations two attributes individually seem to be good candidates to be pruned but if we combine them, they become informative in the structure of tree so there is no need to remove them.

There are two different operations for doing post-pruning: subtree replacement and subtree raising. When we run the algorithm at each node we decide whether to do the subtree replacement, subtree raising, or leave the subtree without any change, unpruned.

The idea in Subtree replacement is to select a subtree and replace it with single leaf. subtree raising - Figure 4.3 is to replace an internal node with one of the nodes below it. As you can see in the example the node C is raised to subsume node B. New child nodes are assigned to node C which will cover all the leaves of node B and C in previous tree.

Figure 4.3: Example of subtree raising [28]

For making the decision about which operation is better at the internal node we should estimate the error rate by the chosen testing set. The error rate will be calculated for leaf nodes too. We can compare these estimations to decide whether to replace, or raise the subtree.

But the testing set may not be available; in this case we use part of the training set. We compute the error at a node by considering the set of instances that reach that node and assume that the majority class in that set is the correct class for that node. The number of misclassified instances over the total number of instances is the error rate for that node.

This error rate and the true probability of error at node is used to find the confidence limit for each node which is used as error rate to make the decision to choose the operation for that specific node. The default value of this factor in C4.5 is 25% and it can be changed to a lower value to have more pruning and it may produce a better result in some cases [28].

In chapter 6 we mention it as confidence factor and by changing we
will see the effect of it to change the structure of the model and recall and precision of test results.

**Number of instances per leaf**

The other parameter for creating a tree with different structure is the minimum number of instances that is allowed to be in a leaf [28]. The default value for this minimum is 2, but we can increase for the cases that have a lot of noisy data. It also could be decreased with datasets that are not noisy but the number of instances for a class is a small number. This will increase the depth of the tree and it may need more pruning in the next step.

### 4.2 Feature vectors

In this section we discuss the way we prepare the data for classification. The method is a supervised classification because we have the labeled data with the name of the classes and also how the data will be represented in our system.

#### 4.2.1 Supervised classification

We need to classify our instances into a discrete set of classes before creating the model. So if we assume that the data set for training instances is $E = \{E_1, E_2, \ldots, E_m\}$ then each training instance $E_j \in E$ is associated with a specific class. More formally, each training instance is a tuple $E_j = (\vec{x}_j, y_j)$, where $\vec{x}_j \in \mathbb{R}^n$ is a vector of features’ values, sampled from a domain $D$, and $y_j \in Y$ is the class to which $\vec{x}_j$ belongs.

The classifier is a function $c : \mathbb{R}^n \rightarrow Y$ which maps the unseen data $\vec{x}$ from testing set which they also should be from the same domain, to values in $Y$ which is the known classes. The main goal of classification is to find this function [18].

#### 4.2.2 Classification features

Vector $(f_1, f_2, \ldots, f_N)$ where $i = 1, \ldots, N$ and $N$ is the total number of attributes is representing each instance whether in training set or testing set. It is the vector $\vec{x}$ that is used as an input to the classifier (function) in previous part.

Here is the function for the vector for given instance of concept $C$:

$$f_i = \begin{cases} 0, & \text{if } p_i \text{ is not associated to } C \\ 1, & \text{if } p_i \text{ is associated to } C \end{cases}$$
4.3 Weka

Weka is a Java suite for machine learning algorithms for data-mining. We can use the program alone and apply the algorithm to the dataset or call the functions inside our Java code. Weka has tools for different tasks like data pre-processing, classification, regression, clustering, association rules, and visualization. [27]

A data set as an input for the Weka algorithms could be as simple as a two-dimensional spreadsheet or database table. The table consists of a number of attributes of which one should be the class value for that instance. The attributes could be in different data types as it is mentioned in [28], like nominal (= one of a predefined list of values), numeric (= a real or integer number) or a string (= an arbitrary long list of characters, enclosed in "double quotes").

4.3.1 ARFF Files

The file type for saving the information in Weka is Attribute-Relation File Format (ARFF). In Table 4.1 you can see the features or attributes of class MA_0001113 with their nominal values:

<table>
<thead>
<tr>
<th>abdomen</th>
<th>cecum</th>
<th>dentin</th>
<th>forelimb</th>
<th>hypopharynx</th>
<th>liver</th>
<th>molar</th>
<th>patella</th>
<th>lung</th>
<th>mouth</th>
<th>...</th>
<th>saliva</th>
<th>thorax</th>
<th>vulva</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1: Part of data table for MA_0001113 class "meninges" concept in Mouse Anatomy

In Figure 4.4, the ARFF file using the features we have in Table 4.1 is created. The relation section contains a name we assign to the ARFF file and it should be a unique, which here is the class code MA_0001113.

The attribute section is the attributes’ names. The data section contains the value of attributes as the feature vectors described in section 4.2.2 in addition with the class attribute for each instance added at the end of vector, because it is a supervised classification.
4.3. WEKA

CHAPTER 4. DECISION TREES IN WEKA

4.3.2 Classifier output

The J48 class which is a decision tree classifier in WEKA is derived from the abstract Classifier class. The main function that is used from it is buildClassifier, which builds the model from training dataset. We use it in Algorithm 3-Build classifier.

This model is a mapping from all-but-one dataset attributes to the class attribute. The difference between this classifier and other type of classifiers is the specific form and creation of this mapping.

Figure 4.6 is an example for J48 classifier of Weka which shows the full output for example file in Figure 4.4 as testing set and with the same format for the training set with same classes and 19 instances and 14 attributes.

At the beginning is a summary of the dataset and test mode which is about the fact that testing set with 5 instances was used to evaluate it. Then a pruned decision tree in textual form is shown. The first split is on the mouth attribute, and then, at the second level, the splits are on thorax and lip, respectively. In the tree structure, a colon introduces the class label that has been assigned to a particular leaf, followed by the number of instances that reach that leaf, expressed as a decimal number because of the way the algorithm uses fractional instances to handle missing values. If there were incorrectly classified instances (for example fovea_centralis) their number would appear, too: thus 4.0/2.0 means that four instances reached that leaf, of which two are classified incorrectly.
Beneath the tree structure the number of leaves is printed; then the total number of nodes (Size of the tree). There is a way to view decision trees more graphically too (Figure 4.5).

![Figure 4.5: Visualize tree for J48 classifier example](image)

The next part of the output gives estimates of the tree’s predictive performance. As you can see, more than 80% of the instances (4 out of 5) have been misclassified. Observe that 1 instance of class meninges has been correctly assigned to this class in model (TP Rate=0.2 or 1 out of 5) and 2 instances have been incorrectly assigned to class fovea_centralis (FP Rate=0.4), one to class of lip and one to of class splenic_artery (FP Rate = 0.2 for both of them).

This is one small example for what a real model in our system will look like and what the result is in details. Some of the measures that are used in this example like True and False positive rates will be discussed in chapter 6.
Figure 4.6: J48 classifier output for part of 4-classes Mouse Anatomy
Chapter 5

Decision Tree matcher

In this chapter we explain how we implement a decision tree matcher for
the ontology alignment framework, SAMBO. The training and testing set
as mentioned before are the corpus documents that are created by querying
the concepts names of input ontologies from PubMed.

The matcher first creates the classifier’s tree or model based on the in-
stances of training set and then by evaluating the instances of testing set
by the model calculates the similarity value between different concepts of
ontologies.

5.1 Method

As it said in section 4.2.2 the main goal of classification is to find a function
or model for the set of data and by using this function predict the value of
specific attribute (class attribute) for unseen data instances. In other words
we classify or match the unseen data to known set of classes so the function
can be called matcher or classifier.

We create the feature vector for each downloaded document for each
concept of ontologies from PubMed (max number of downloaded document
is 15). Instances are changed from abstract’s words to the word bag which
contains the words that are repeated more than defined threshold (for ex-
ample repeated more than 10 times in the whole process will be done over
the words inside the documents’ abstract) and they are the result of apply-
ing the removal of unwanted information. Then they will be stored in the
database for further access because the program could not always have all
the data in the heap during the run time.

Classification features as described in 4.2.2 will be created from the result
of previous step. These features are detected from both ontologies because
we need to have the same structure for feature vectors for both training
and testing set instances. The class attribute should be assigned to each
instance according to section 4.2.1, which is the name of concept related to
5.2 Implementation steps

The steps of building a decision tree matcher are as follow:

- Create corpus (Algorithm 1): download desired number of abstracts per concept for both ontologies from PubMED.

Figure 5.1: Algorithm 1- Build Corpus from PubMed PubMedCorpus-Builder.buildCorpus

- Create feature vector and database (Algorithm 2)
  - Create the feature vector as described in previous section. As it said before the test and training set should contain same amount of attributes in other word they should be compatible.
  - save the feature vector and feature list in the database

- Build the classifier (Algorithm 3): query the feature from database for the training ontology for all the concepts, set the options like Min number of instances per leaf, confidence factor and create the model for it. Create the ARFF file for each concept
  - Without the concept’s Parent: querying the feature vector from database for that specific concept.
  - With the concept’s Parent: the query will contain the parent feature vectors also which will be labeled as the child class.
5.2. IMPLEMENTATION

CHAPTER 5. DECISION TREE MATCHER

Figure 5.2: Algorithm 2-Create feature vectors from corpus DecisionTree-BuildDatabase.generateCorpusDB

Use this model for creating the C4.5 classifier. It will contain all concepts in training set. Then by Evaluation the classifier against the testing instances the class that has the true positive value will be selected as the suggested concept for the testing concept.

For some concepts the true positive value for none of the classes is greater than zero so by having the greatest false positive value after applying a threshold the new suggestion will be produced (explained in section 4.1.2).

This step should be repeated for the testing dataset to use the average of similarity value of both results to get symmetric result.

• Evaluate the testing instances (Algorithm 4): first we calculate the number of pairs which are correctly chosen in algorithm and reference list (a), the ones which are not in reference list but chosen as suggested pair by algorithm (b), the ones are in the reference list and not in the suggestion pairs by algorithm (c) and other cases (d), which can be seen in Table 5.1 - Suggestion matrix:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Real</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>correct</td>
<td>a</td>
</tr>
<tr>
<td>wrong</td>
<td>correct</td>
<td>c</td>
</tr>
</tbody>
</table>

Table 5.1: Suggestion matrix

Then we calculate the 3 evaluation measures:
5.2. IMPLEMENTATION

CHAPTER 5. DECISION TREE MATCHER

Figure 5.3: Algorithm 3-Build classifier

```
require Ontology1, ontology2, parent, threshold, MaxNumberOfInstancesPerLeaf, ConfidenceFactor

// Create arffFile for ontology
BufferedWriter.write( relation_name -> ontology1.name )
// Retrieve all features from features table
for all feature in features do
    BufferedWriter.write( attribute -> feature.name +{value domain} [0,1] )
end for

// Retrieve all featureVectors from ontology1 table
for all featureVector in ontology1 do
    BufferedWriter.write( data -> featureVector +ontology1.classNames )
end for

instances1 <- arffFile
ontology1.setClassIndex(instances1.indexOf("concept_class") )
// Set Classifier Parameters: MaxNumberOfInstancesPerLeaf, ConfidenceFactor
J48.setMinNumObj(MaxNumberOfInstancesPerLeaf)
J48.setConfidenceFactor(ConfidenceFactor)
model <- J48.buildClassifier(instances1)

// Create arffFile for class
BufferedWriter.write( relation_name -> class.name )
// Retrieve all features from features table
for all feature in features do
    BufferedWriter.write( attribute -> feature.name +{value domain} [0,1] )
end for

// Retrieve all featureVectors from ontology1 table
for all featureVector in class do
    BufferedWriter.write( data -> featureVector +class.name )
end for

if parent then
    for all featureVector in parentClass do
        BufferedWriter.write( data -> featureVector +parentClass.name )
    end for

instances2 <- arffFile
ontology2.setClassIndex(instances2.indexOf("concept_class") )
eval <- Evaluation(instances1)
model <- J48.evaluateModel(model, instances2)
if rates.truePositive > 0 then
    suggestion <- model.class, similarityValue = 1
else if rates.falsePositive > threshold then
    suggestion <- model.class, similarityValue = rates.falsePositive

// Save class.name, suggestion in simValues table
```

1. The Precision is the proportion of the examples which truly have class $x$ among all those which were classified as class $x$.

$$Precision = \frac{a}{a + b}$$

2. How much part of the class was captured is equivalent to Recall

$$Recall = \frac{a}{a + c}$$

3. The F-Measure is a combined measure for precision and recall which is computed like:

$$F-measure = \frac{(precision \times recall \times 2)}{(precision + recall)}$$

Evaluation by reference alignment is a traditional evaluation method which is building a reference alignment manually and measures the
5.3 Implementation

The SAMBO suite is developed in Java, we download the lastest version of Weka [22] and add the Weka suite as a jar file to the existing libraries and use the class J48 for building C4.5 classifier from the feature vectors and Evaluation to evaluate the testing set via the created model by classifier [23], [26].

In Appendix A: Implemented Functions Description you can see the list of implemented classes and functions of the algorithms 1-4. This list also contains classes that are used for connecting to SQL Server database and also loading the ontologies and reference alignment.

5.4 Limitations

One of the limitations was the size of the feature vector which was produced by the abstract’s text key words. The model for classifier was made by this and when the size of the model file was more than the heap size the program crashed.

Also because of size of the feature vector the instances for training and testing set could not be queried from the database, so the ARFF was created one by one for each concept in the step before creating the classifier.

We are using the QueryPubMED inside the SAMBO system which uses Field Tag [Majr], and in this case some of the concepts will not have any instances then no result from the created model will be assigned to them, so we tried with Field Tag [All fields] which solved this problem but was so
inefficient in time (approximately 145 hours running time compared to 2-10 hours for other runs of algorithm). In further work chapter there are some suggestions for solving these limitations.
Chapter 6

Experiment

6.1 Basic concepts

6.1.1 Dataset

As we mentioned in chapter 2 about the alignment framework of SAMBO, the matchers need two ontologies as input. The ontologies we use for the tests are Human and Mouse ontologies from anatomy track of OAEI 2011 [13].

They are the datasets for Ontology Alignment Evaluation Initiative Campaign’s datasets with the reference alignment which were used for evaluating the result. They are structured as directed acyclic graphs with is-a and part-of relationships.

6.1.2 Measures definition

True and false positives

The True Positive (TP) rate is the number of instances classified as class x over all instances which truly have class x. The classified classes of testing set which have the True Positive value more than zero will be introduced as the suggestion with similarity value 1.

The False Positive (FP) rate is the number of instances classified as class x, but belong to a different class, over all instances which are not of class x. The classified classes of testing set which do not have the True Positive value more than zero will be assigned the highest FP rate in the list of classes of model as a pair suggestion for them with similarity value equal to FP rate. This is because although they belong to another class but if the FP rate is high it means the attributes have a lot in common with the class it is wrongly classified with and it may be the correct class for suggesting. This approach improves the recall for the test result. Having the FP rate value
helps to control the amount of these suggestions by applying the threshold which also will improve the precision.

**Threshold**

Threshold is the limit that we use to accept the suggestion with similarity value above it and remove the rest of the suggestions. In most of the cases it improves the precision, but sometimes it also removes the correct suggestions with small similarity value with has a bad effect on recall.

Finding the best value for threshold is not our goal in this paper because it depends on the collection of instances [2] but in general the effect of using it will be discussed later. The range of values we use for threshold are 0.2, 0.4, 0.6, 0.8 and 1. With threshold equal to one we will have just the suggestions with TP rate value more than zero.

### 6.1.3 Classifier attributes

Table 6.1 shows the options for J4.8 classifier. For example you can force the algorithm to use the unpruned tree instead of the pruned one. You can avoid subtree raising, which increases efficiency. You can set the confidence threshold for pruning and the minimum number of instances at any leaf.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>-U</td>
<td>Use unpruned tree</td>
</tr>
<tr>
<td>-C</td>
<td>Specify confidence threshold for pruning</td>
</tr>
<tr>
<td>-M</td>
<td>Specify minimum number of instances in any leaf</td>
</tr>
<tr>
<td>-R</td>
<td>Use reduced-error pruning</td>
</tr>
<tr>
<td>-N</td>
<td>Specify number of folds for reduced-error pruning; use one fold as pruning set</td>
</tr>
<tr>
<td>-B</td>
<td>Use binary splits only</td>
</tr>
<tr>
<td>-S</td>
<td>Do not perform subtree raising</td>
</tr>
<tr>
<td>-L</td>
<td>Retain instance information</td>
</tr>
<tr>
<td>-A</td>
<td>Smooth the probability estimates using Laplace smoothing</td>
</tr>
<tr>
<td>-Q</td>
<td>Seed for shuffling data</td>
</tr>
</tbody>
</table>

Table 6.1: Scheme-specific options for the J4.8 decision tree learner

Some of these options were tested in our algorithm which will be introduced here:

**Number of instances per leaf:**

The C4.5 has this capability to have the min number of instances per leaf as 0 or more. The test cases have been done with setting this parameter to 1, 2 and 3 (explained in section 4.1.3).
Confidence factor

For pruning the classifier’s tree we need to set the confidence factor. The smaller the value means more pruning in the model tree (Explained in section 4.1.3). Here the four different values 0.05, 0.15, 0.25 and 0.35 were used.

6.1.4 Document based options

Corpus size

Max numbers of abstract per concept of ontology are 5, 10 and 15 documents and also with the whole corpus or part of corpus according to selected searching methods (chapter 3). Having more instances to classifying a concept or evaluating a test instance against the model will be examined here.

Structure-Based informations

One of the information about the concept that we can use from ontologies sources is relations between concepts like is-a relations. This information could be used during the alignment process by assuming that abstracts that are related to the sub-concepts of a concept class are also related to that concept [19].

In the algorithm, we select the feature vectors of concepts along with the feature vectors of their parents with the class attribute equal to sub-concept’s class. It gives the sub-concepts more instances to be examined or classified by.

6.1.5 Combination with other classifiers

The suggestions can be determined based on the similarity value from one matcher, or the combination of the similarity values measured by several matchers using weights.

For given concept C1 in ontology 1 and Concept C2 in ontology 2 we define:

\[ \text{sim}(C1, C2) = \frac{\sum_{k=1}^{n} w_k \times \text{sim}_k(C1, C2)}{\sum_{k=1}^{n} w_k}, \]

Where \( \text{sim}_k \) and \( w_k \) represent the similarities and weights, respectively, for the different matchers (combination in Figure 2.4).

The results were combined with text-based classifiers like Ngram, TermBasic, TermWN, UMLSKSearch, BayesLearning. We set the weight for both sides 50% and accept the suggestions with similarity value above the threshold 0.4 (Figure 6.1).
6.2 Analysis

6.2.1 Test result for DT classification-whole corpus

There are different options that can be set for C4.5 matcher inside the Weka implementation. Besides that there is variation which produced two set of results which were not related to properties of the matcher. There is an option in querying PubMed which was described in Algorithm 1. If we do-Query[All fields] it will create the corpus for all the concepts in ontologies and as the result the model will be much more bigger. Building the classifier and evaluating the result in this case took a huge amount of time (approximately 145 hours compared to 2-10 hours for other runs of algorithm) and repeating it for different set of attributes was not possible, so there is just one set of results in Table 6.2 for just one running of algorithm and in the next section the result will be just with the doQuery[Major fields] which is mentioned as efficient version and uses just part of corpus. It was with the cost of decreasing in the recall.

![Figure 6.1: Combination and filtering](image.png)

Table 6.2: Result of classification with whole corpus

<table>
<thead>
<tr>
<th>Whole corpus with site5</th>
<th>Confidence factor</th>
<th>a,b,c/th</th>
<th>precision</th>
<th>recall</th>
<th>f-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min number of instances per leaf=2</td>
<td>Without parents</td>
<td>0.25</td>
<td>836,686,680/0.2</td>
<td>0.549</td>
<td>0.551</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>834,630,682/0.4</td>
<td>0.570</td>
<td>0.550</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>797,169,719/0.6</td>
<td>0.625</td>
<td>0.526</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>789,75,727/0.8</td>
<td>0.913</td>
<td>0.520</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>775,58,741/1.0</td>
<td>0.930</td>
<td>0.511</td>
</tr>
</tbody>
</table>

As you can see in Table 6.2, the average number of correct suggestions (=806) in comparison with same variable (average a=256) in Table 6.3 is more than three times which has a huge effect on recall but the number of incorrect suggestions is larger which will effect precision.

Figure 6.2 also shows that the recall is always higher when we use the whole corpus but it will decrease the precision when we compare its value with the same threshold in test result for part of the corpus and the whole corpus. But in general the f-measure in test result related to whole corpus are higher than 50 percent while in part of corpus test results are less than
50 percent.

Figure 6.2: Comparison of classification with whole corpus and part of corpus

![Comparison of classification with whole corpus and part of corpus]

### 6.2.2 Test result for DT classification- part of corpus:

Table 6.3 is the result of test runs of program with different options for C4.5 matcher and just part of the downloaded corpus.

The number of correct suggestions (means value of a in the table) has a small variation from 236-261 and mostly around the average 256 so the recall is not changing that much, but different threshold value causes the number of incorrect suggestions to increase and make the precision worse.

In some of the next parts we just discuss the effect of the different settings of attributes only about precision because the recall value does not change that often.
6.2.3 Comparison with other classifiers

There are other text-based matchers like NGram, TermBasic and TermWN, domain knowledge based like UMLSKSearch and also instance-based like BayesLearning in SAMBO and we can compare the result of their mappings with our results with the threshold value equal to 0.8 which you can see in Figure 6.3.

The precision in both instance-based matchers is equal to 1 but the recall
6.2. ANALYSIS

**CHAPTER 6. EXPERIMENT**

Figure 6.3: Comparison between C4.5 and other matchers with threshold = 0.8

<table>
<thead>
<tr>
<th>Matcher</th>
<th>Recall</th>
<th>Precision</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>BayesLearning</td>
<td>0.038</td>
<td>1.000</td>
<td>0.074</td>
</tr>
<tr>
<td>C4.5-part of corpus</td>
<td>0.171</td>
<td>1.000</td>
<td>0.292</td>
</tr>
<tr>
<td>N-Gram</td>
<td>0.706</td>
<td>0.619</td>
<td>0.659</td>
</tr>
<tr>
<td>TermBasic</td>
<td>0.625</td>
<td>0.997</td>
<td>0.768</td>
</tr>
<tr>
<td>TermWN</td>
<td>0.625</td>
<td>0.997</td>
<td>0.768</td>
</tr>
<tr>
<td>UMLSKEarth</td>
<td>0.819</td>
<td>0.871</td>
<td>0.844</td>
</tr>
</tbody>
</table>

in our algorithm result is higher (more than 4 times). The f-measure and recall in text-based matchers is higher but the precision is lower.

6.2.4 Combination with other classifiers

We can combine different matchers and compute the similarity value like it was described in section 5.1.7. The Figure 6.4 and Figure 6.5 show the result of C4.5 whole corpus/part of corpus matcher and the other matchers alone and with combination with C4.5.

The weight of combination for both matchers is 50% with the threshold (0.4) for the computed similarity value and the matchers also.
6.2. ANALYSIS

CHAPTER 6. EXPERIMENT

Figure 6.4: C4.5 part of corpus compared to other matchers with & without combination

![Graph](image1)

In both Figures we can see the recall decreased while precision increased in most of the cases when we used the combination of other matcher with our result, except the Bayes Learning combination result which was reverse when we used the whole corpus.

6.2.5 Parent concepts and minimum number of instances per leaf

When we use the parent’s concept instances in addition to the concept instances, the result mostly was different with different min number of in-
stances per leaf and it could be seen more in the lower thresholds.

For example when we set min number of instances per leaf=0 with threshold = 0.2 the result of using parents’ concepts has higher precision while with threshold = 0.4 and same min number of instances per leaf the result of using parents’ concepts has lower precision compared to not using parents’ concepts.

Table 6.4: Precision for different Parent concepts and min number of instances per leaf

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Using parents</th>
<th>min number of instances per leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Without parents</td>
<td>With parents</td>
</tr>
<tr>
<td>0.2</td>
<td>0.893</td>
<td>0.897</td>
</tr>
<tr>
<td></td>
<td>0.900</td>
<td>0.896</td>
</tr>
<tr>
<td>0.4</td>
<td>0.942</td>
<td>0.935</td>
</tr>
<tr>
<td></td>
<td>0.942</td>
<td>0.935</td>
</tr>
<tr>
<td>0.6</td>
<td>0.985</td>
<td>0.981</td>
</tr>
<tr>
<td></td>
<td>0.974</td>
<td>0.974</td>
</tr>
<tr>
<td>0.8</td>
<td>0.996</td>
<td>0.996</td>
</tr>
<tr>
<td></td>
<td>0.996</td>
<td>0.996</td>
</tr>
<tr>
<td>1.0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

But in general decreasing the number of instances in leaf in the model tree improves the precision.

Figure 6.6: Using parents’ instances with different min number of instances per leaf

6.2.6 Confidence factor

Increasing the confidence factor until 0.25 makes the precision and f-measure better without any effect on recall, but increasing the value of it to 0.35
decreases the precision. It means the amount of limit of pruning for the tree model equal to 0.25 is the best value could be set and it mostly depend on the structure of input data.

<table>
<thead>
<tr>
<th>Confidence factor</th>
<th>Min number of instances per leaf=0</th>
<th>Min number of instances per leaf=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.864</td>
<td>0.860</td>
</tr>
<tr>
<td>0.15</td>
<td>0.881</td>
<td>0.875</td>
</tr>
<tr>
<td>0.25</td>
<td>0.887</td>
<td>0.878</td>
</tr>
<tr>
<td>0.35</td>
<td>0.881</td>
<td>0.872</td>
</tr>
</tbody>
</table>

Table 6.5: Compare precision for different confidence factor

It shows that depending on the instances we use for the tree different settings could cause different result as it was in the previous section too.

6.2.7 Threshold and corpus size

The precision improved by having more instances per class, but not a significant improvement in recall. It means the algorithm could choose better suggestion when more information about the class is available. Recall just improved when Lower threshold and smaller corpus size is used together.

<table>
<thead>
<tr>
<th>Corpus size threshold</th>
<th>05 precision</th>
<th>recall</th>
<th>10 precision</th>
<th>recall</th>
<th>15 precision</th>
<th>recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.8764</td>
<td>0.1701</td>
<td>0.8966</td>
<td>0.1715</td>
<td>0.8993</td>
<td>0.1715</td>
</tr>
<tr>
<td>0.4</td>
<td>0.8784</td>
<td>0.1708</td>
<td>0.9353</td>
<td>0.1715</td>
<td>0.9384</td>
<td>0.1715</td>
</tr>
<tr>
<td>0.6</td>
<td>0.9810</td>
<td>0.1708</td>
<td>0.9831</td>
<td>0.1708</td>
<td>0.9885</td>
<td>0.1702</td>
</tr>
<tr>
<td>0.8</td>
<td>0.9961</td>
<td>0.1708</td>
<td>0.9962</td>
<td>0.1708</td>
<td>1.0000</td>
<td>0.1702</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0000</td>
<td>0.1708</td>
<td>1.0000</td>
<td>0.1708</td>
<td>1.0000</td>
<td>0.1702</td>
</tr>
</tbody>
</table>

Table 6.6: Effect of threshold on different size of corpus
6.3  Conclusions

As you can see in the result we have the highest f-measure when we have larger corpus size and smaller min number of instances per leaf with the same value for the confidence factor (=0.25).

The highest precision is when we use the higher threshold with the instances without taking account their parent concepts in the testing set.

We have the best result for the recall when we used the lowest threshold (=0.2) with 10 max documents in the corpus.
Chapter 7

Discussions

7.1 Conclusion

In this paper we introduced the implementation of a decision tree matcher for ontology alignment for biomedical ontologies in the SAMBO system. The test result for different setting is diverse and in some cases it depends on the instances we choose to represent our classes from the ontologies. But in general we see that the confidence factor for pruning the tree which was built by our classifier from our nominal data was independent from the data and the best was 0.25 as it is the default value in Weka system too. The other setting which is the min number of instances per leaf for the decision tree creates different result when we set the threshold to different values. In general using higher threshold helped to have higher precision in most cases because it rolls out the wrong suggestions with small similarity value.

We also evaluated the influence of the size of the literature corpus, the quality and performance of the strategies and their combination with other strategies. Using the whole corpus produced the much better result in recall but it has an inefficient run time as we discussed so we mostly used the part of the corpus for running the tests.

Our algorithm outperforms the text-based matchers in most cases with higher precision, although compared to the other matchers in SAMBO the recall is lower. We could use this as an advantage and by combining our result with the other matchers’ results, get more precise suggestions and remove the wrong pairs from the suggested list.

7.2 Further work

The algorithm we used here used one model based on the all instances we found related to the ontologies’ concepts which was not so efficient in time, using some other algorithm to produce a better result like dividing the model
into several models or producing the instances in parallel could be some good improvement.

Another change in order to make some improvement in the result could be using another algorithm to extract the bag of words from abstracts of PubMed result to make more proper and smaller feature vectors. One suggestion could be to get an efficient feature vector by using document classification in Weka and selecting the best attributes which are selected from the bag of words for the feature vector.
Bibliography


### Appendix A

## Implemented Functions

<table>
<thead>
<tr>
<th>Description</th>
<th>Class Name</th>
<th>DecisionTreeMatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Private Functions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>setClassifierParameter(withParent, threshold, instancesPerLeaf, confidenceFactor, corpusSize)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getFeatures()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getCategories(tablename)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getConceptClass()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getClasses(tablename1, tablename2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getModel(training_data)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>insertSimConcepts(term1, term2, similarityValue)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>buildARFF(term, getParents, ontology, tablename, corpusDir, classes, isModel)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getSuggested(isCombination, weight1, weight2, threshold, matcher)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>evaluateStrategies(isCombination)</td>
<td></td>
</tr>
<tr>
<td><strong>Public Functions</strong></td>
<td>BuildClassifier()</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getSimValue(Element1, Element2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>getSimValue(term1, term2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>main(String[] args)</td>
<td></td>
</tr>
<tr>
<td><strong>Private Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Public Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Package Name</strong></td>
<td></td>
<td>se.liu.ida.sambo.component.matcher.learning.decisiontree</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td></td>
<td>This class creates the parameters which is needed to make the J48 classifier in Weka and then create one and evaluate the result.</td>
</tr>
</tbody>
</table>
# Appendix A. Implemented Functions Description

<table>
<thead>
<tr>
<th>Class Name</th>
<th>DecisionTreeBuildDatabase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Functions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ListDirectoryInstances(dirName)</td>
</tr>
<tr>
<td></td>
<td>ListFileInstances(dirName, maxAbstractNumber)</td>
</tr>
<tr>
<td></td>
<td>getAbstract(fileName)</td>
</tr>
<tr>
<td></td>
<td>getFeatures(fileName)</td>
</tr>
<tr>
<td></td>
<td>CreateTable tablename</td>
</tr>
<tr>
<td></td>
<td>InsertINTODB(pid, conceptid, name, feature, parents)</td>
</tr>
<tr>
<td></td>
<td>generateFeatures(corpusPath1, corpusPath2, maxAbstractNumber)</td>
</tr>
<tr>
<td>Public Functions</td>
<td>generateCorpusDB(ontology1, category1, ontology2, category2, corpusDir, maxAbstractNumber)</td>
</tr>
<tr>
<td>Private Variables</td>
<td></td>
</tr>
<tr>
<td>Public Variables</td>
<td></td>
</tr>
<tr>
<td>Package Name</td>
<td>se.liu.ida.sambo.component.matcher.learning.decisiontree</td>
</tr>
<tr>
<td>Description</td>
<td>This class creates the database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Name</th>
<th>PubMedCorpusBuilder</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Functions</td>
<td></td>
</tr>
<tr>
<td>Public Functions</td>
<td>buildCorpus (ontoClasses, ontoName, maxAbstractNumber, corpusRootDir)</td>
</tr>
<tr>
<td>Private Variables</td>
<td></td>
</tr>
<tr>
<td>Public Variables</td>
<td></td>
</tr>
<tr>
<td>Package Name</td>
<td>se.liu.ida.sambo.component.matcher.learning.Bayers.PubMedCorpusBuilder</td>
</tr>
<tr>
<td>Description</td>
<td>builds the corpus by downloading abstracts/articles from PubMed</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Name</th>
<th>OntoLoader</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Functions</td>
<td></td>
</tr>
<tr>
<td>Public Functions</td>
<td>loadOntology (urlSet, ontoId, withReason)</td>
</tr>
<tr>
<td></td>
<td>removeObsoleteClasses()</td>
</tr>
<tr>
<td>Private Variables</td>
<td></td>
</tr>
<tr>
<td>Public Variables</td>
<td></td>
</tr>
<tr>
<td>Package Name</td>
<td>se.liu.ida.sambo.component.matcher.learning.decisiontree</td>
</tr>
<tr>
<td>Description</td>
<td>Load the ontology file as a MOntology object which will contain the concept names, relations, synonyms.</td>
</tr>
</tbody>
</table>
### Class Name: extractRA

#### Private Functions
- openFile()
- extractData(String[] input)
- getRA()

#### Public Functions
- openFile()
- extractData(String[] input)
- getRA()

#### Private Variables
- ArrayList RA

#### Public Variables
- ArrayList RA

#### Package Name
- se.liu.ida.sambo.component.matcher.learning.decisiontree

#### Description
- Load the reference alignment as an array list

---

### Class Name: SQLServer

#### Private Functions

#### Public Functions
- SQLServer(username, password, url)
- finalize()
- executeUpdate(SQLQuery)
- executeQuery(SQLQuery)

#### Private Variables

#### Public Variables

#### Package Name
- se.liu.ida.sambo.component.matcher.learning.decisiontree

#### Description
- Creates a new instance of SQL and create the connection to the database.
Appendix B

Setup Guide for the System

SAMBO desktop application, capable of aligning medium-size or large-size ontologies. It contains

- SAMBO_DeskApp, a desktop application with similar user interface as SAMBO web application. It is capable of running ordinary alignment process with single threshold filtering.

- RunAlignmentProcess, a java program (without user interface) capable of running advanced alignment process, such as double threshold filtering (i.e. SAMBOdtf [4]) and most PRA algorithms in [1] (i.e. fPRA, dtfPRA, mgPRA and mgfPRA.)

- PubMedCorpusBuilder, a java program (without user interface) for downloading PubMed corpus for a given set of terms or ontological concepts

- BayesMatcher, a java program (without user interface) for building and using BayesMatcher for aligning ontologies

B.1 Setup SAMBO-DeskApp Project in NetBeans IDE

1. Open NetBeans IDE and load in the project in the "Project Folder"

2. Configure WordNet
   (a) Install WordNet 2.1 for Windows.
      (Available at http://wordnet.princeton.edu/wordnet/download/)
B.2 Run SAMBO-DeskApp application (without IDE)

1. Install Java and JDK, setup the Java environment variables "JAVA_HOME" and "JDK_HOME"
2. The project should have been built in IDE. If not, "Clean and Build" the project in the IDE.
3. Configure WordNet
   (a) Install WordNet 2.1 for Windows.
       (Available at http://wordnet.princeton.edu/wordnet/download/)
   (b) In the Files window, open the file "/config/wordnet.xml", and set "dictionary_path" to the directory of WordNet dictionary files
       (e.g. param name="dictionary_path" value="Y:/WordNet-2.1/dict")
4. The application SAMBO_DeskApp can be run by executing "run.bat" file in the "Project Folder"

B.3 Run the matchers

1. TermBasic matcher
2. TermWN matcher
3. UMLSKSearch matcher
   Note: This matcher will query the domain knowledge on UMLS knowledge source server, which will cost some time. During running, you can check its status and outputs from Tomcat terminal.
4. The Hierarchy matcher
   Note: This matcher is designed to do similarity propagation based on already known mappings. If you do not have given mappings at the beginning, do not use this matcher during the first run.
5. BayesLearning matcher
   Note: This matcher requires a corpus of documents and generates text classifiers. This is a time consuming matcher that should
only be used after having tried the other matchers. According to our experience, it should also not be used on its own, but in combination with other matchers.

B.4 To inspect the code

1. We recommend using NetBeans 6.9 with JDK 6.0 or later version to load in SAMBO project.
2. Before running it, we need to configure WordNet path similar as before.

NOTES:
For libraries, there might be reference problem. To resolve this, re-import all libraries.
Appendix C

Nomenclature

OWL Web Ontology Language
NCBI National Center for Biotechnology Information
PMID PubMed Identifier
MEDLINE Medical Literature Analysis and Retrieval System Online
NLM National Library of Medicine
E-Utilities Entrez Utilities
OAEI Ontology Alignment Evaluation Initiative
MA Mouse Anatomy
NCI National Cancer Institute
MeSH Medical Subject Headings
SAMBO System for Aligning and Merging Biomedical Ontologies
Weka Waikato Environment for Knowledge Analysis
ID3 Iterative Dichotomiser 3
ARFF Attribute Relationship File Format
IDE Integrated Development Environment
Using ontologies is a key technology in the semantic web. The semantic web helps people to store their data on the web, build vocabularies, and has written rules for handling these data and also helps the search engines to distinguish between the information they want to access in web easier. In order to use multiple ontologies created by different experts we need matchers to find the similar concepts in them to use it to merge these ontologies.

Text based searches use the string similarity functions to find the equivalent concepts inside ontologies using their names. This is the method that is used in lexical matchers. But a global standard for naming the concepts in different research area does not exist or has not been used. The same name may refer to different concepts while different names may describe the same concept.

To solve this problem we can use another approach for calculating the similarity value between concepts which is used in structural and constraint-based matchers. It uses relations between concepts, synonyms and other information that are stored in the ontologies. Another category for matchers is instance-based that uses additional information like documents related to the concepts of ontologies, the corpus, to calculate the similarity value for the concepts.

Decision trees in the area of data mining are used for different kind of classification for different purposes. Using decision trees in an instance-based matcher is the main concept of this thesis. The results of this implemented matcher using the C4.5 algorithm are discussed. The matcher is also compared to other matchers. It also is used for combination with other matchers to get a better result.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Uphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English
The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/
© Tahereh Boujari