MEASURING NAMED ENTITY SIMILARITY THROUGH
WIKIPEDIA CATEGORY HIERARCHIES

by

JARED M ASHMAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2010

ACKNOWLEDGEMENTS

I would like to thank Dr. Chengkai Li, my thesis advisor, without whom I could not have completed this thesis. Under his supervision and assistance, I have learned a great deal about how to perform research. Dr. Li has helped me to grow as a student, especially in the way I approach and solve problems. He has also taught me how, when and where to focus my efforts, which has served me well many times while researching and writing this thesis.

Dr. Bahram Khalili, my graduate advisor, also deserves thanks and recognition, for all of his help and guidance throughout my journey in the masters program at the University of Texas at Arlington. He has helped me accomplish more in the last years than I thought possible.

Finally, I would like to thank my wife, Heather, and my family, Gabriel, Alexandra, Joshua and Nicholas, whose daily love and support have more than once kept me moving towards completion of this thesis. My wife has always acted as my sounding board and has helped me come to the right decision more times than I could ever count. She inspires me through her example. I love you Heather.

November 19, 2010

ABSTRACT

MEASURING NAMED ENTITY SIMILARITY THROUGH WIKIPEDIA CATEGORY HIERARCHIES

Jared M Ashman, M.S.

The University of Texas at Arlington, 2010

Supervising Professor: Chengkai Li

Identifying the semantic similarity between named entities has many applications in NLP, including information extraction and retrieval, word sense disambiguation, text summarization and type classification. Similarity between named entities or terms is commonly determined using a taxonomy based approach, but the limited scalability of existing taxonomies has led recent research to use Wikipedia's encyclopedic knowledge base to find similarity or relatedness. These existing methods using Wikipedia have so far focused on relatedness, but are not as well suited to finding similarity. In this thesis, we evaluate methods for determining the semantic similarity between named entities by associating each named entity to a specific Wikipedia article, and then using the commonalities between Wikipedia category hierarchies as the similarity. To evaluate the effectiveness, we conducted a survey to get manually defined similarity scores for named entity pairs. The scores obtained were then compared to both implemented methods and existing relatedness measures.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS....................iii

ABSTRACT....................iv

LIST OF ILLUSTRATIONS....................viii

LIST OF TABLES....................ix

Chapter Page

1 INTRODUCTION....................1

1.1 Semantic Similarity and Semantic Relatedness....................1

1.1.1 Semantic Similarity Versus Semantic Relatedness....................2

1.2 Methods of Determining Semantic Similarity or Semantic Relatedness....................2

1.2.1 Wikipedia as a Taxonomy....................3

1.2.2 Mapping of Concepts....................3

1.3 Named Entities....................4

1.4 Overview of Our Methods: CatSim....................5

1.4.1 Wikipedia Category Hierarchy....................5

1.4.2 Intuition Behind CatSim....................6

1.5 Overview of Our Results....................7

1.6 Rest of the Thesis....................7

2 RELATED WORK....................8

2.1 Wikipedia Based Semantic Similarity Measures....................8

2.2 Wikipedia Based Semantic Relatedness Measures....................9

3 APPROACH....................11

3.1 Basics....................11

3.1.1 Wikipedia Category Hierarchy Challenges....11

3.2 Collapsing the Category Hierarchy....12

3.3 Vector Comparison Methods....15

3.3.1 Dice’s Coefficient....15

3.3.2 Jaccard Coefficient....17

3.3.3 Cosine Similarity....18

3.4 CatSim Example....19

4 IMPLEMENTATION....22

4.1 Wikipedia Database....22

4.1.1 Preprocessing....22

4.2 CatSim....23

4.2.1 SQLAlchemy....23

4.2.2 List Comprehensions....24

5 RESULTS....25

5.1 Relatedness and Similarity Evaluation Metrics....25

5.2 Relatedness and Similarity Surveys....25

5.2.1 SimSurvey....25

5.2.1.1 Named Entity Pair Selection....25

5.2.1.2 Observations on Respondent Data....26

5.3 Experimental Results....26

5.3.1 Effects of Different Vector Comparison Methods....27

5.3.2 Effects of Different Weighting Functions....27

5.3.3 Effects of Different Depths....27

5.4 Relatedness Results....28

5.5 Similarity Results....28

5.6 Overall Impression....29

6 FUTURE WORK....33

6.1 Improving the Weighting Function....33

6.1.1 TF-IDF Weighting Function....33

6.2 Vector Comparison Methods Revisited....33

6.3 Category Collapse Algorithmic Improvements....34

6.4 Efficiency Optimizations....34

6.5 Extension From Named Entities to Concepts....34

6.5.1 Wikipedia Disambiguation Pages....35

6.5.2 Wikipedia Link References....35

6.6 Real World Test Applications....35

REFERENCES....37

BIOGRAPHICAL INFORMATION....39

LIST OF ILLUSTRATIONS

Figure Page

1.1 Measures of Semantic Relatedness and Semantic Similarity....2

1.2 Wikipedia Category Hierarchy....5

1.3 Comparing Category Hierarchies....6

3.1 Partial “Tiger” Category Hierarchy....11

3.2 Collapsing the Hierarchy....13

3.3 Category Collapse Algorithm....14

3.4 Dice’s Coefficient....15

3.5 Dice Based Algorithm....16

3.6 Jaccard Coefficient....17

3.7 Jaccard Based Algorithm....17

3.8 Cosine Similarity....18

3.9 Cosine Similarity Based Algorithm....19

3.10 Comparing Category Hierarchies Revisited....20

4.1 Euclidean Distance Demonstrating List Comprehensions....24

5.1 Dice’s Coefficient: WordSim353....30

5.2 Dice’s Coefficient: SimSurvey....30

5.3 Jaccard Coefficient: WordSim353....31

5.4 Jaccard Coefficient: SimSurvey....31

5.5 Cosine Similarity: WordSim353....32

5.6 Cosine Similarity: SimSurvey....32

LIST OF TABLES

Table Page

2.1 Correlation of Wikipedia Based Relatedness Measures to WordSim353.................................9

5.1 Relatedness Correlation Results to WordSim353...28

5.2 Similarity Correlation Results to SimSurvey..29

CHAPTER 1

INTRODUCTION

1.1 Semantic Similarity and Semantic Relatedness

Semantic similarity and semantic relatedness are often used synonymously. Both similarity and relatedness are measures of how close two concepts are to one another. Natural language processing (NLP) takes advantage of the measures for many applications, including information extraction and retrieval, word sense disambiguation, text summarization, and type classification. However, these applications typically use similarity and relatedness interchangeably, which has led current research to focus mainly on semantic relatedness, when semantic similarity may be better than semantic relatedness or vice versa, as they are different.

Specifically, semantic similarity is a subset of semantic relatedness. Similarity includes hyponymic and hypernymic relationships (*is-a*), while relatedness includes any and all functional relationships (*has-a*, *is-a-part-of*, etc.)[12]. The differences lead to several observations when determining a semantic similarity or semantic relatedness measure for a pair of concepts. Figure 1.1 illustrates the observations on how a concept pair falls into one of four areas with respect to the pair's measure of relatedness and similarity.

Area (1) on Figure 1.1 is not possible for a concept pair to fall into. Since similarity is a subset of relatedness, it is impossible to have a high similarity measure and a low relatedness measure. Where similarity and relatedness measures are both high, concept pairs like “George Washington” / “Abraham Lincoln” or “tiger” / “jaguar” fall firmly into area (2). The opposite holds for (3), where similarity and relatedness measures should both be low, like in the concept pair “nirvana” / “cheese grater”. Both these areas give credence to the idea that similarity and relatedness are synonymous, as the measures are roughly equivalent with each other.

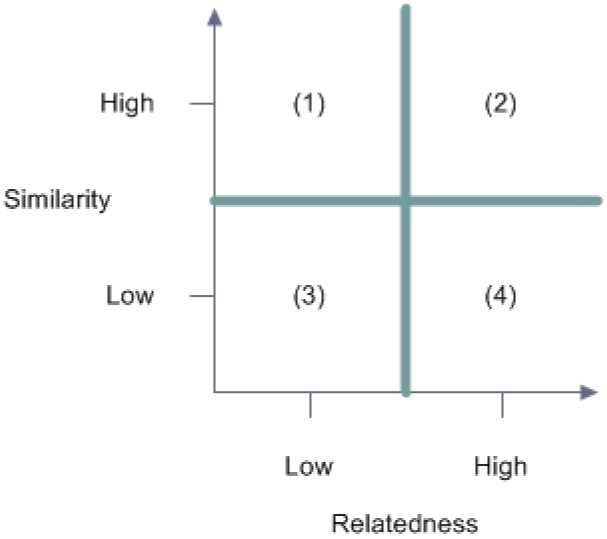


Figure 1.1 Measures of Semantic Relatedness and Semantic Similarity

However, there is a large class of concept pairs, like “Maradona” / “soccer” and “OPEC” / “oil” that fall into area (4). These concept pairs have a high relatedness, but low similarity. In any of the cases that fall into (4), semantic similarity and semantic relatedness are quite clearly not synonymous and the majority of conflict between the two measures is to be found.

1.1.1 Semantic Similarity Versus Semantic Relatedness

Due to the difference detailed above, on most data sets of concept pairs, methods created to measure semantic relatedness should not perform well when measuring semantic similarity, though these methods often use semantic similarity and semantic relatedness as synonyms. To date, as far as we have been able to determine, there has been no direct correlation comparison between semantic similarity methods and semantic relatedness methods when measuring both semantic similarity and semantic relatedness.

1.2 Methods of Determining Semantic Similarity or Semantic Relatedness

Traditionally, organized and well defined taxonomies such as WordNet[5] are used to classify concepts and determine the relatedness or similarity measure for a concept pair. What these traditional taxonomies lack is large coverage and scale. The taxonomies do not contain esoteric or recent concepts, and do not scale quickly as it takes a prohibitive amount of time to add concepts to the taxonomy[8].

1.2.1 Wikipedia as a Taxonomy

Wikipedia[13], in contrast, contains over 3.4 million articles in the English version alone as of November 2010. Wikipedia, like WordNet and other taxonomies, also contains internal structure and classification. These structures include the categories that an article belongs to as well as any internal links to other Wikipedia articles that are in the article text. Being open for editing by anyone, Wikipedia grows incredibly quickly, and contains both esoteric and recent articles and concepts.

Being a collaborative and open effort always means that Wikipedia has up to date information, but this means that the structure is not as well defined as a controlled taxonomy. This ill defined structure introduces variability that needs to be accounted for. The variability could be as simple as a mistake in classifying an article into a category, or may be as severe as deliberate vandalism, and in either case can affect the similarity or relatedness measure.

1.2.2 Mapping of Concepts

In all of the methods examined in this thesis document, Wikipedia is used as the taxonomy for determining the semantic similarity or semantic relatedness measurement. Since in NLP applications, the measurement is between two concept pairs, and Wikipedia consists of articles, the concepts must be mapped to Wikipedia. This usually means that the concept is equated with a single article from Wikipedia that best matches the context between the concept in the pair. Some methods map the concepts automatically, and some use a manual mapping. Regardless of whether it is done manually or automatically, there are three types of mapping that take place.

First is a single direct correspondence, where a concept directly corresponds to an article in Wikipedia. An example of this mapping would be to map the concept “car” to Wikipedia. Wikipedia does not have an article named “Car”, but it does have an “Automobile” article that the concept directly corresponds to.

Second, a concept may apply to more than one article in Wikipedia, and may have to be disambiguated based on the context. In this case, the context is the other concept in the pair, for example, the concept “king” could refer to the Wikipedia article for “Monarch” or “King_(chess)” depending on if the other concept in the pair was “country” or “rook” respectively.

Finally, a concept may not apply directly to any article in Wikipedia, and has to be mapped to an article in Wikipedia that is similar or encapsulates the concept. This would be the case for the concept “string”, which does not have an article in Wikipedia. The concept is encapsulated by the article “Rope”.

Manually mapping concepts to articles involves searching Wikipedia for the concept. Once searched, the appropriate article can be selected depending on what type of mapping is called for and what the concept is.

While time consuming, the human judgments involved in manually mapping concepts to Wikipedia articles is typically very accurate. Automatic mapping of the concept to the article is much more prone to mistakes. Some methods using the Wikipedia taxonomy that have been tried are to use the most frequently linked article for the concept[8], or to use Wikipedia’s disambiguation pages to determine which pair of articles has the best similarity score[12].

The methods described in this thesis do a manual mapping to a Wikipedia article, and thus lend themselves well to measuring the similarity of named entities rather than being applicable to concepts as a whole.

1.3 Named Entities

Each article in Wikipedia usually describes a named entity. Named entities are sequences of words that identify an entity. Examples of named entities are “Barack Obama”, “Golden Gate Bridge” and “tiger”. Each named entity falls into a particular type in a named entity hierarchy such as the BBN’s Proposed Answer Categories for Question Answering[2]. These types are broad categories like “person”, “location” or “animal.” When mapping a concept to Wikipedia, a particular article or articles are selected as applicable for the concept, so the method is distilling a concept down to named entities.

1.4 Overview of Our Methods: CatSim

A method of determining semantic similarity measures between two concepts that have already been mapped to Wikipedia articles is introduced in this thesis: CatSim. CatSim works in two stages: first by creating a weighted vector by collapsing the category hierarchies of the mapped articles, then proceeding to perform a configurable vector comparison between the vectors, yielding a similarity measurement score for the concepts. The implemented vector comparisons are well known in information retrieval literature, Dice's Coefficient, Jaccard's Coefficient and cosine similarity.

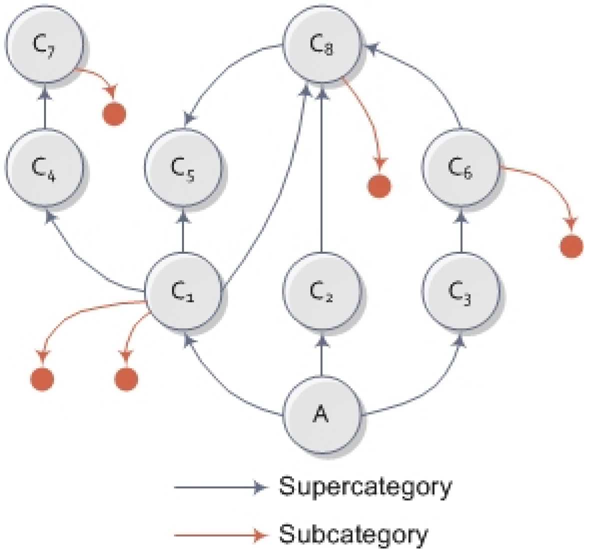


Figure 1.2 Wikipedia Category Hierarchy

1.4.1 Wikipedia Category Hierarchy

The Wikipedia category system forms a type of taxonomy called a folksonomy, where the collaborative tagging of articles into different categories both categorizes and provides means to connect articles together. Any given category is allowed to have one or more subcategories or supercategories (Figure 1.2), which forms a hierarchy of categories. The articles may be assigned to any and all categories. This creates a very rich taxonomy to mine relationship data from.

1.4.2 Intuition Behind CatSim

In CatSim, the category hierarchies of two articles are compared together to determine a similarity measurement (Figure 1.3). When two articles have categories in common, even supercategories up several levels, it follows that the articles are similar, as they have been tagged to belong to the same category hierarchies. This forms the basis on which the different comparison algorithms implemented for CatSim operate. The more categories that two articles have in common, the greater the similarity.

To determine the similarity measure, CatSim first constructs a category vector for each of the articles containing all of the categories that are in that article's category hierarchy, and then performs a standard vector comparison method on the two resultant category vectors. The vector comparison methods chosen to be implemented in this thesis are well known in information retrieval and data mining, Dice's Coefficient, Jaccard's Coefficient, and the cosine angle similarity measurement.

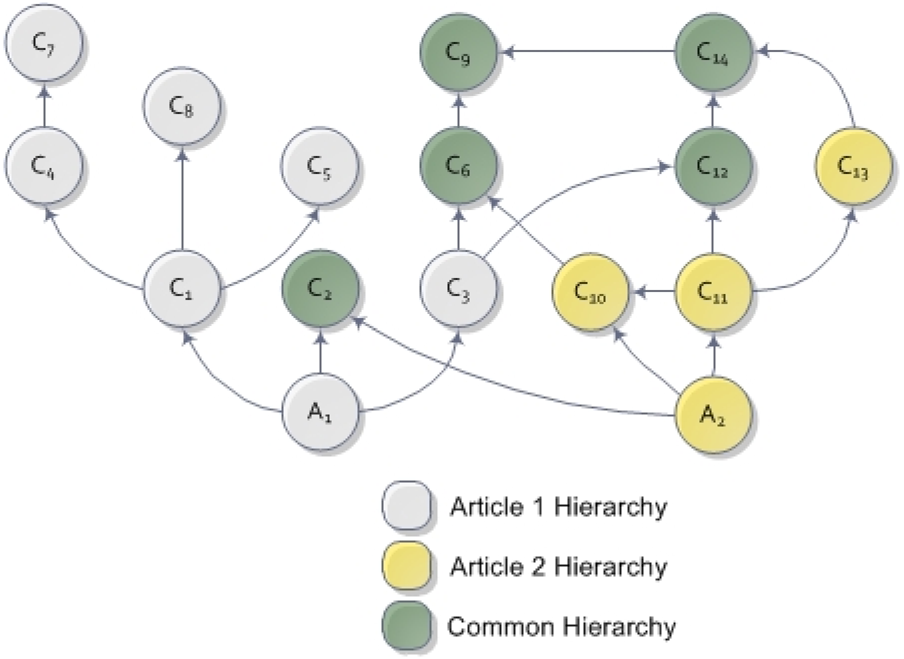


Figure 1.3 Comparing Category Hierarchies

1.5 Overview of Our Results

Many of the similarity and relatedness techniques use the same evaluation methods. One common method is to compare the technique in question to human judgments. First compute the results of the technique for a large number of concept or named entity pairs. Once that has been done, determine the correlation coefficient of those results to those of human measurements for the same concept or named entity pairs. There is an existing study, WordSim353[6], that has determined the relatedness between concept pairs, but no such study was found for similarity. Therefore we constructed a set of 80 named entity pairs, and conducted a survey to determine the similarity between each of the named entity pairs.

Experimental results on CatSim against the relatedness based WordSim353 test data set and the similarity survey showed the following:

i. CatSim performed significantly better than existing relatedness methods when correlating to the human judgments in the similarity survey.

ii. CatSim performed on par with other existing Wikipedia based relatedness measures, but not as well as the state of the art when correlating to the WordSim353 test data set.

iii. While varying the weighting function and the comparison method used affected the correlation to the test data for both relatedness and similarity, in most cases it was not a significant amount.

iv. Varying the maximum category depth increased the correlation to the test data much more than varying the other parameters.

1.6 Rest of the Thesis

In the rest of the thesis, we first present in chapter 2 an overview of the related work for semantic similarity and semantic relatedness that use Wikipedia. Then in chapter 3 we will cover in detail the semantic similarity measurement methods developed in this thesis. Chapter 4 dives into our implementation of the semantic similarity measurement methods. Experiments on the implemented similarity methods and comparisons with existing relatedness methods comprise chapter 5. The thesis document wraps up in chapter 6 with future directions that will be pursued.

CHAPTER 2

RELATED WORK

2.1 Wikipedia Based Semantic Similarity Measures

Much existing literature for named entities focuses on semantic similarity between named entities as a method to enhance the identification of named entities from text. This task is called Named Entity Recognition (NER) and was formally described in the 6th Message Understanding Conference in 1995. One or the original types of named entities in NER is person names. To identify and disambiguate person names, Bunescu and Pasca[3] utilized Wikipedia's article text and the categories that articles belonged to in order to drive a support vector machine (SVM). The taxonomy based kernel for the SVM took as an input a concept, then looked at that concept's text to identify and disambiguate to a set of possible articles. Finally it identified context based information contained in the possible articles' text and categories that matched the ambiguous concept. This method of using the category hierarchy and article text to find the similarity measurement between the concept and possible articles achieved an 84% disambiguation accuracy when applied to a disambiguation task.

Cucerzan[4] also used Wikipedia article text and category pages to create a system that used semantic similarity for entity identification and disambiguation. The method employed by Cucerzan was to process the contextual information contained in the concept, then match the context of candidate articles and their category information in order to maximize the agreement between the article context and the concept context. The most similar article to the concept context was selected as the disambiguated article. Cucerzan obtained accuracy results of 88% to 91% on disambiguation tasks dealing with named entities.

Bollegala et. al.[1] approached determining semantic similarity in a different way. Instead of using Wikipedia, they used another web based massively scalable knowledge base: search engines. Bollegala focused on determining the correlation coefficient between their web search methods and human similarity results. To determine the semantic similarity between two entities, they performed three queries, one for the first term, one for the second term, then one for the first and second term joined together. The resultant queries contain snippets of text that are then mined for syntactic patterns to determine the similarity. The correlation coefficient between their web query and snippet pattern based approach to the Miller and Charles[9] 30 word pair sample data set was 0.834.

2.2 Wikipedia Based Semantic Relatedness Measures

When expanding the scope to include relatedness as well as similarity, the number of different implementations of different methods to determine semantic relatedness between concepts or named entities greatly increases. Restricting it to just those methods that directly deal with Wikipedia yields three methods: Strube and Ponzetto's WikiRelate![12], Gabrilovich and Markovitch's Explicit Semantic Analysis (ESA)[7] and Milne and Witten's Wikipedia Link-based Measure (WLM)[8]. The three methods' correlation coefficients to the WordSim353 data set are detailed in Table 2.1.

Table 2.1 Correlation of Wikipedia Based Relatedness Measures to WordSim353

Relatedness Measure	Correlation Coefficient
WikiRelate!	0.48
Explicit Semantic Analysis	0.75
Wikipedia Link-based Measure	0.68

Strube and Ponzetto's WikiRelate! was the first to explore the use of Wikipedia to determine semantic relatedness. They examined path based, information content based, and text overlap based relatedness measurement methods. From the experimentation, the path based measures had the best correlation coefficient at 0.48 to the WordSim353 data set.

The path based measures developed compute a relatedness measure based on the shortest path through the category hierarchy between two articles. The maximum allowed path of the search was limited since all articles go through the base categories in the Wikipedia category hierarchy.

After Strube established the use of Wikipedia as a taxonomy for semantic relatedness measures, Gabrilovich and Markovitch developed ESA, which markedly improved the correlation coefficient to the WordSim353 data set to 0.75. ESA uses a cosine similarity measure over entity vectors to establish a relatedness measure. The entity vectors consist of every article that mentions the concepts being compared, and are built up using an inverted index.

While achieving state of the art accuracy, ESA requires an enormous amount of preprocessing. Milne and Witten attempt to address this with WLM. WLM works on the internal hyperlinks within Wikipedia articles to determine the relatedness measure.

WLM averages two relatedness measurements to get the final relatedness measure. First WLM computes the cosine similarity between TF-IDF weighted vectors of two articles' outgoing links to other Wikipedia articles. The second measurement is a term occurrence based measurement of all of the articles that contain a link to the two articles being compared. This combination approach of incoming and outgoing Wikipedia links gives good relatedness measurements with WLM achieving a correlation coefficient of 0.68 with the WordSim353 data set.

CHAPTER 3

APPROACH

3.1 Basics

Like WikiRelate!, ESA and WLM, our approach to measuring semantic similarity, CatSim uses Wikipedia's vast knowledge base. Specifically, we are focusing on named entities that have already been mapped to Wikipedia articles and the category hierarchy of those articles to determine the semantic similarity of the named entities. The intuition is that articles that are similar to one another tend to have categories in their individual hierarchies in common (Figure 1.3). CatSim takes advantage of this intuition by using a two step method to first collapse each article's category hierarchy into a vector, and second comparing the vectors together to get a similarity score.

3.1.1 Wikipedia Category Hierarchy Challenges

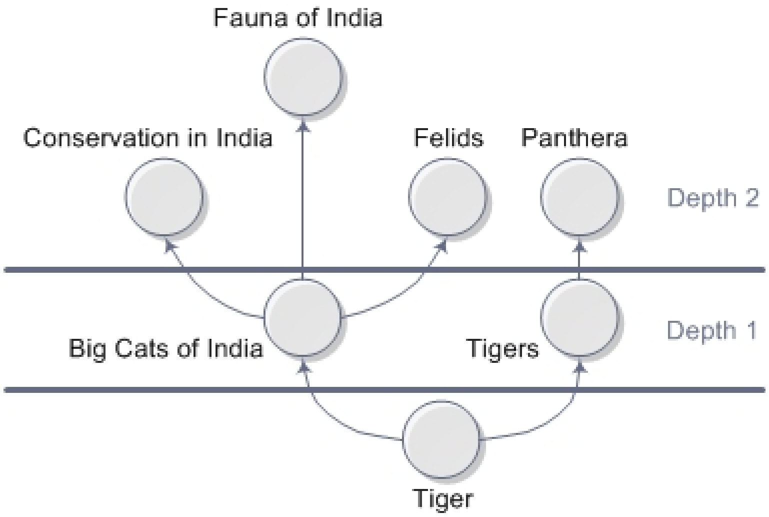


Figure 3.1 Partial “Tiger” Category Hierarchy

A challenge to using the Wikipedia category hierarchy lies in the structure of the hierarchy itself. Described briefly in Chapter 1, The hierarchy is a tree structure with a single root category, but is free form and contains multiple inheritance, huge branching factors and cycles. While this creates a rich information source to mine, it also introduces difficulties in processing it.

Every article in Wikipedia is classified into one or more categories, and can act as it's own root for an article based category hierarchy. The article rooted category hierarchy can be constructed by starting with all of the article's categories, then branching from each of those categories to their supercategories. The example above in Figure 3.1 illustrates creating a category hierarchy, take the "Tiger" article in Wikipedia. "Tiger" is a member of the "Big cats of India" and "Tigers" categories at a depth of 1. The "Big Cats of India" category has the categories "Conservation in India," "Fauna of India" and "Felids" as supercategories at depth 2. The "Tigers" category has a single supercategory "Panthera," which, since it's a super category of a depth 1 category, is depth 2.

3.2 Collapsing the Category Hierarchy

Conceptually, to compare two article's category hierarchies, there has to be a way to measure the overlap between them. CatSim accomplishes this through collapsing the category hierarchy for the article down into a category vector. The category vector contains the categories that represent all of the unique categories that exist in the article's category hierarchy, each mapped to a weighted value for the category.

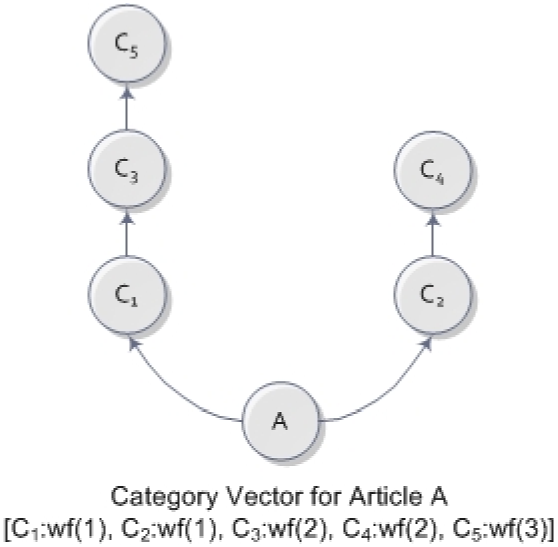

C_5
C_3
C_4
C_1
C_2
A
Category Vector for Article A
[C_1:wf(1), C_2:wf(1), C_3:wf(2), C_4:wf(2), C_5:wf(3)]

Figure 3.2 Collapsing the Hierarchy

Figure 3.2 conceptualizes the method of collapsing the hierarchy, showing an article, A and a simplified hierarchy with categories C_1...C_5. The hierarchy is moved into a vector one category at a time, applying a weight function (wf(d) in Figure 3.2), to each to get a weighted value for the category to place into the category vector. The weight function can be any function that operates on the depth of the category in the category hierarchy.

```
Algorithm: collapse(base:Page, depth:int, maxdepth:int):list

Input:
    base: Article or Category to get the categories for
    depth: The current depth the algorithm is working at
    maxdepth: The maximum depth the algorithm will traverse to

Output:
    cv: Category vector of the category hierarchy of the list type

begin

    if depth <= maxdepth
        for each c in categories-of(base)
            if c not in cv
                add-to-cv(c, wf(depth)))
            else
                if wf(depth) > weight-of(cv[c])
                    replace-in-cv(c, wf(depth)))
            collapse(c, depth + 1, maxdepth)
    else
        return

end
```

Figure 3.3 Category Collapse Algorithm

The specific algorithm used to collapse an article’s categories is shown in Figure 3.3. The recursive algorithm is invoked through a call to collapse(base, depth), In the initial call, base is the original item to work on, either the base article, or a category within the base article’s category hierarchy. Two further variables come into play in the category collapse algorithm. First, the algorithm allows a depth and a maximum depth to be specified, the maximum depth is the maximum path length from the article to a category through other categories. The greater the depth, the higher in the overall Wikipedia category hierarchy and the more often the category will appear in any article’s hierarchy. The categories near the root of the overall Wikipedia category hierarchy are also general in nature, and thus are not as helpful in determining similarity. For these reasons, it makes sense to limit the article’s category hierarchy to a specified maximum depth. Depth is the current depth the algorithm is working on and since the algorithm is recursive acts as the end condition. Depth is initially called with a value of 1.

Second is the weighting function, wf(d). This function is applied to every category as it is placed into the category vector, weighting that category. The variable d is the shortest path between that category and the base article. Broad classes of weighting functions can be used, giving different shapes to the weighting depending on the curve of the function used.

Notice that the category vector includes unique categories. Since Wikipedia allows multiple inheritance and cycles, there exists the possibility that a category is listed twice in an article hierarchy. In this case, it is necessary to select a single instance of the category to place into the category vector. We chose to use the instance of the category that has the largest weight. Intuitively, it makes sense that having a higher weight for the category lends a more accurage measure of similarity, and so the larger weight is selected.

3.3 Vector Comparison Methods

Once a weighted category vector has been created for both of the target articles, the vectors must be compared. There are a large number of vector comparison methods in the literature. Several of the most commonly used comparison methods seemed to have desirable features. Three were selected for inclusion in the initial implementation of CatSim. Dice's Coefficient, Jaccard Coefficient and the standard cosine similarity.

3.3.1 Dice's Coefficient

$$D(A,B)=2\frac{|A\cap B|}{|A|+|B|}$$

Figure 3.4 Dice's Coefficient

Dice's Coefficient is a simple set based similarity measure that gives a stronger emphasis to common categories in the category vectors due to doubling the weight of the intersection. This emphasis while comparing gives a wider range of scores when differing weights and counts of common categories are found. The formula above, Figure 3.4, works on the input A and B, which in this case, as well as in Figure 3.6 and 3.8, are the two category vectors being compared.

```
Algorithm: dice(a1:Article, a2:Article, maxdepth:int):float

Input:
    a1: Article to perform the dice vector comparison for
    a2: Article to perform the dice vector comparison for
    maxdepth: The maximum depth the algorithm will traverse to

Output:
    m: Dice measurement of the similarity between the two articles

begin

1   cv1 = collapse(a1, 1, maxdepth)
2   cv2 = collapse(a2, 1, maxdepth)
3   1u2 = cv1 union cv2
4   for each c in 1u2
5       if c in cv1
6           add-to-ucv1(c, weight-of(cv1[c])
7       else
8           add-to-ucv1(c, 0)
9       if c in cv2
10          add-to-ucv2(c, weight-of(cv2[c])
11      else
12          add-to-ucv2(c, 0)
13      if c in cv1 and cv2
14          add-to-1i2(c, (weight-of(cv1[c] + weight-of(cv2)[c]) / 2)
15      else
16          add-to-1i2(c, 0)
17  return (2 * sum-of-values(1i2)) / (sum-of-values(ucv1) + sum-of-values(ucv2))

end
```

Figure 3.5 Dice Based Algorithm

The Dice based algorithm in Figure 3.5 compares two category vectors together and returns a similarity measure. In the algorithm, the meat of it is in lines 4 through 16, which is where the union category vectors and the intersection vector is created, which are ucv1, ucv2 and 1i2 respectively. The add-to-ucv1 (lined 6 and 8), add-to-ucv2 (lines 10 and 12) and add-to-1i2 (lines 14 and 16) methods simply add a category associated with a weight to a category vector.

A point of interest is that the intersection logic takes the average of the weighted values of the categories, rather than picking the highest or lowest. Taking the average insures that both category vectors have equal representation in the intersection vector. The intersection vector also contains as many values as the union, any categories that are not in both category vectors are inserted into the intersection vector with a value of 0, and thus contribute nothing to the similarity measurement, but do affect the size of the intersection vector.

Once the intersection is calculated, computing the Dice similarity measurement is a simple division of the summed intersection vector over the addition of the summation of each category vector.

3.3.2 Jaccard Coefficient

$$J(A,B)=\frac{|A\cap B|}{|A\cup B|}$$

Figure 3.6 Jaccard Coefficient

The Jaccard Coefficient is much like Dice's Coefficient. The major difference is that it does not offer any emphasis to common categories, note that Jaccard's coefficient does not multiply the intersection by 2. This should produce a more even distribution of measurements given the wide range of weights and number of common categories.

```
Algorithm: jaccard(a1:Article, a2:Article, maxdepth:int):float

Input:
    a1: Article to perform the dice vector comparison for
    a2: Article to perform the dice vector comparison for
    maxdepth: The maximum depth the algorithm will traverse to

Output:
    m: Jaccard measurement of the similarity between the two articles

begin

    cv1 = collapse(a1, 1, maxdepth)
    cv2 = collapse(a2, 1, maxdepth)
    1u2 = cv1 union cv2
    for each c in 1u2
        if c in cv1 and cv2
            add-to-1i2(c, (weight-of(cv1[c] + weight-of(cv2)[c]) / 2)
        else
            add-to-1i2(c, 0)
    return sum-of-values(1i2) / sum-of-values(1u2)

end
```

Figure 3.7 Jaccard Based Algorithm

The Jaccard based algorithm in Figure 3.7 starts exactly the same as the Dice based algorithm, gathering the union vector. When gathering the union vector, it operates a little differently than the intersection vector described above in section 3.3.1. When categories that are in common between the two category vectors are found, the union operation will take the largest weighted value found in the category vectors for that category.

The Jaccard based algorithm is also easy to compute once the intersection vector has been created. Since we already have the union vector, we return the summation of the intersection vector over the summation of the union vector.

3.3.3 Cosine Similarity

$$C(A,B)=\frac{A\cdot B}{||A||||B||}$$

Figure 3.8 Cosine Similarity

Cosine similarity is used in many applications, perhaps most often in text mining. It works well to find the similarity between TF-IDF vectors. We selected cosine similarity because it was used in other similarity and relatedness applications, both in those that use Wikipedia like ESA and WLM as well as those that use more traditional taxonomies.

The cosine similarity based algorithm in Figure 3.9 below was the most computationally complex algorithm, as the dot product and Euclidean distance for both category vectors needs to be calculated. This creates a much slower overall execution speed than either the Dice or Jaccard methods.

Notice, like the intersection vectors from the Dice and Jaccard algorithms, the individual category vectors are extended out to include the categories from the other category vector. The new categories in the category vectors have a value of 0, and do not affect the dot product or euclidean distance.

Interestingly, due to the extension of either a category vector or the creation of an intersection vector, all three algorithms start out with the same steps, creating a union vector. This is probably not strictly necessary and definitely hurts the execution speed of both Dice's coefficient and the cosine similarity, as neither algorithm needs the union.

```
Algorithm: cosine(a1:Article, a2:Article, maxdepth:int):float

Input:
    a1: Article to perform the dice vector comparison for
    a2: Article to perform the dice vector comparison for
    maxdepth: The maximum depth the algorithm will traverse to

Output:
    m: Cosine similarity measure of the similarity between the two articles

begin

    cv1 = collapse(a1, 1, maxdepth)
    cv2 = collapse(a2, 1, maxdepth)
    1u2 = cv1 union cv2
    for each c in 1u2
        if c in cv1
            add-to-ucv1(c, weight-of(cv1[c])
        else
            add-to-ucv1(c, 0)
        if c in cv2
            add-to-ucv2(c, weight-of(cv2[c])
        else
            add-to-ucv2(c, 0)
    for i in 1..length-of(1u2)
        add-to-dv(weight-of(ucv1[i]) * weight-of(ucv2[i])
    1dotproduct2 = sum-of-values(dv)
    for each c in ucv1
        add-to-euc1(weight-of(ucv1[c])^2)
    euclidean1 = square-root(sum-of-values(euc1))
    for each c in ucv2
        add-to-euc2(weight-of(ucv2[c])^2)
    euclidean2 = square-root(sum-of-values(euc2))
    return 1dotproduct2 / (euclidean1 * euclidean2)

end
```

Figure 3.9 Cosine Similarity Based Algorithm

3.4 CatSim Example

To illustrate the way CatSim performs the two step process, we will start with Figure 3.10, which delineates two articles and their category hierarchies, including common categories. For the example, we will be limiting the category hierarchy to a depth of 4, applying a weighting function of wf(d) = 1/d and using the Jaccard based method for the vector comparison.

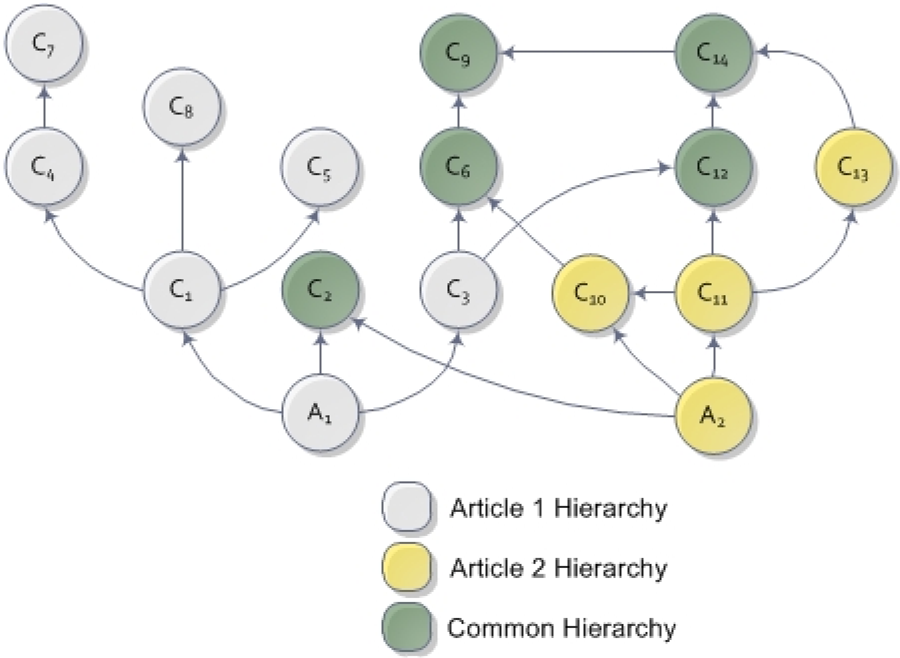


Figure 3.10 Comparing Category Hierarchies Revisited

First, category vectors must be created for both A_1 and A_2. For A_1, starting at depth 1, we have C_1, C_2 and C_3. Depth 2 has C_4, C_8, C_5, C_6, and C_{12}. Depth 3 consists of C_7, C_9 and C_{14}, while the hierarchy ends at depth 4 with C_9. Moving through the depths and assigning weight functions to those categories, A_1's category vector is [C_1:1, C_2:1, C_3:1, C_4:0.5, C_8:0.5, C_5:0.5, C_6:0.5, C_{12}:0.5, C_7:0.3, C_9:0.3, C_{14}:0.3]. C_9 at depth of 4 is not included in the category vector as that category already exists for the hierarchy at a depth of 3. In the same way, the category vector for A_2 is [C_2:1, C_{10}:1, C_{11}:1, C_6:0.5, C_{12}:0.5, C_{13}:0.5, C_{14}:0.3, C_9:0.3].

Once we have our category vectors, apply the Jaccard Coefficient to the category vectors. For Jaccard, we need the intersection of the category vectors divided by the union of the category vectors to get the similarity value. The intersection of the category vectors is the set of all common categories, with an average of their weighted value. The intersection set for our example is [C_2:1, C_6:0.5, C_{12}:0.5, C_9:0.3, C_{14}:0.3]. If we sum the values for the intersection set we get a value of 2.6. To calculation the union of the category vectors, add each of the categories from both article's category vectors to the union set. For categories that exist in both the category vectors, the largest weighted value is taken. In our example the union set is [C_1:1, C_2:1, C_3:1, C_4:0.5, C_8:0.5, C_5:0.5, C_6:0.5, C_{12}:0.5, C_7:0.3, C_9:0.3, C_{14}:0.3, C_{10}:1, C_{11}:1, C_{13}:0.5]. The sum of the values in the union set in this example comes out to be 8.9. Following the Jaccard algorithm and formula, we divide the intersection set value by the union set value, 2.6/8.9, or 0.2921, which is the similarity score using the Jaccard Coefficient for the example hierarchies in Figure 3.10.

CHAPTER 4

IMPLEMENTATION

4.1 Wikipedia Database

The Wikipedia database is at the heart of the implementation. Without it's vast knowledge store, CatSim would not have anything to work on. The version of Wikipedia that CatSim is currently running on a download from March of 2010. Due to the nature of CatSim using only the internal structure of Wikipedia rather than any of the content, it was not necessary to download the entirety of the Wikipedia database. We only had to download the tables that directly dealt with the structures we were interested in: Page, Category, PageLinks, CategoryLinks, and Redirect.

4.1.1 Preprocessing

WikiRelate!, ESA and WLM all have detailed preprocessing that must be followed to provision the Wikipedia data for use. One of the initial goals of CatSim was to create a system that could be set up quickly and easily from a Wikipedia database dump, without needing preprocessing. We did not completely succeed, as we do require one preprocessing step.

Wikipedia's category hierarchy does not just have categories that describe content. It also includes administrative categories such as lists, classes, templates, help and different categories that are used to mark an article as needing cleanup or having problems that need fixed. None of these categories are of use when determining semantic relatedness or similarity, and in fact may hinder efforts at a correct measurement. To account for these administrative categories, a preprocessing step is done to delete them from the system.

A list was created with common text strings contained in the title of the category. This list was used to query the database and delete any categories that contained those strings. Some examples of the administrative category text strings were "-related_", "Wikipedia_", "templates" and "Automatically_assessed_". In case they were necessary later, the category information that was deleted was saved to a new table in the database that was not used during operation of CatSim.

4.2 CatSim

After setting up and preprocessing the database we decided to implement CatSim in the Python programming language. The decision to implement in Python in part was driven by two factors that eventually made things very easy. These factors are the Object Relational Mapper (ORM) SQLAlchemy[11] and a language feature of Python, list comprehensions[10].

4.2.1 SQLAlchemy

In any application that uses a database as a back end, how you access that database can make or break the performance and reliability of the application. A clean solution to database access are ORM solutions. ORM libraries or modules map the relational model in the database to the object model in code. They abstract away the database access so that no SQL is necessary to write, and make it easy to pull the data into objects during runtime. This allows the user to focus not on database access but to strongly focus on the application logic.

SQLAlchemy is one such ORM for the Python programming language. Besides having all of the features of a standard ORM, it includes one feature that is key for CatSim's performance when loading all of the levels of categories. Typically when loading a list of categories and their supercategories, an application would load all of the first level categories, and then for each category in the first level, would load all of the second level categories. This would continue until the maximum depth was reached. In practice, this results in a database query and extra processing time for each category. This common problem causes an enormous degradation in performance.

The SQLAlchemy feature we used to get around this query problem and solve the performance issue allows you to override the set of loaded instances of the supercategories. When combined with a dictionary structure to map which super category belonged to which category, it reduces the number of queries against the database to the maximum depth allowed between the article and categories, greatly improving performance.

Accepting an article to load, the article load method's key point is that it works on a depth basis, acting on all of the categories for the article that are at that depth. It will recursively travel up, loading the next level's data, mapping the returned data to the category in the current level, and setting each level as processed so that it does not load again. Each category will be loaded only once, as the existence of loaded instances of supercategories is checked before issuing the call to the database.

Note that this algorithm can be implemented without the use of an ORM, but the ORM makes this much easier overall.

4.2.2 List Comprehensions

The Python programming language has a lot of excellent features, but by far the most useful for working with vectors and sets as we do in CatSim are list comprehensions. List comprehensions work by creating a list or set from another list or set based on evaluation statements. An example from CatSim would be calculating the Euclidean distance of a category vector for the cosine similarity vector comparer (Figure 4.1).

```
distance = sqrt(sum([v*v for v in categoryvector]))
```

Figure 4.1 Euclidean Distance Demonstrating List Comprehensions

The bold area in the figure is the list comprehension. The list comprehension iterates over every value in the category vector, and squares the value. The list comprehension then constructs a new list with all of those squared values and returns it. The built in sum function and the math sqrt function are then applied to quickly and easily get the Euclidean distance.

CHAPTER 5

RESULTS

5.1 Relatedness and Similarity Evaluation Metrics

Existing relatedness measures that use Wikipedia have consistently used the same metrics to evaluate how accurate the measure is. This metric is performing Pearson's correlation coefficient over the rank values between the WordSim353 test data set and the relatedness measures obtained by the method being evaluated. This gives a value between 0 and 1 that shows how correlated the WordSim353 rankings are to the relatedness method's rankings. Table 2.1 described in Chapter 2 above shows the existing Wikipedia based methods. A value of 0 means that it is as uncorrelated as possible, while a value of 1 means that the accuracy is perfect, the relatedness method being tested has exactly the same rankings as the human judgments in WordSim353.

5.2 Relatedness and Similarity Surveys

While WordSim353 deals with relatedness data, we could not find an appropriate equivalent measure for semantic similarity that highlighted the difference between relatedness and similarity. As a result, we conducted our own similarity survey to create such a data set (SimSurvey).

5.2.1 SimSurvey

SimSurvey consists of 81 named entity pairs that have clear mappings to Wikipedia articles for easy processing with the CatSim application. The survey gathered 12 human respondents, with each having all 81 pairs evaluated.

5.2.1.1 Named Entity Pair Selection

To determine which named entity pairs would be included, a broad spectrum of different classifications were considered. Since named entities have a type associated with them, and named entity pairs that consist of the same type should be similar, we selected named entity types to focus on. The types included were: person, facility, organization, NORP (nationality, other, religion, and political), geo-political entity, location, plant, animal and game. Within each type, we selected named entities that would have high name recognition for a person located in the United States.

The pairs were then selected within and across the types to fall into 3 groups and one sub group. We expected 30 of our pairs to have high similarity, 25 to have a middling similarity and 26 to have low similarity. Of the 26 low similarity named entity pairs, 6 of those selected were of high relatedness, such as the named entity pair “Michael Jordan” / “basketball.”

Respondents were then asked to assign a similarity score to each of the 81 named entity pairs, on a score from 0 to 10, with 0 being no similarity and 10 being the same named entity. All respondent scores were averaged at the end to get an overall score for each named entity pair.

5.2.1.2 Observations on Respondent Data

The respondents to the survey classified the data as expected, especially the entity pairs that were of low similarity and high relatedness. There was some blending, as some of the high similarity items tended towards the middle of the score range, such as “Judaism” / “Catholic”. This entity pair contains two named entities that are both religions and have a lot of similar properties, though the respondents only scored the pair at a 6.3333, where we felt it should have been much higher.

5.3 Experimental Results

Figure 5.1 through Figure 5.6 detail out the experimental results that were run on the CatSim application. We varied the vector comparison methods, weighting function, depth, and test data set to correlate against to determine the most effective combination of parameters. Regardless of whether the test data set was the WordSim353 for relatedness or SimSurvey for similarity, we noticed some overall trends in the data.

5.3.1 Effects of Different Vector Comparison Methods

Varying the vector comparison methods had moderate effect on the overall data. Within each test data set's results, the vector comparison methods all seemed to be grouped within 0.05 of each other when the other parameters remained the same.

Both the Dice based comparison and Jaccard based comparison had similar measurements, so the Dice based comparison that emphasized the intersecting categories more did not actually have much effect on the overall measurement scores.

5.3.2 Effects of Different Weighting Functions

Like varying the vector comparison methods, we varied the weighting functions used. We wanted to see what the effect of different function classes would have on the correlation coefficient between the results and the test data sets. The intuition says that if you use a function that weights categories that are farther from the article less, you will achieve a better similarity measurement score.

The intuition proved sound to a point. We measured the effects of exponential, polynomial, linear and constant functions, from both growing and shrinking perspectives based on level. Based on the data obtained, yes, weighting the categories less based on larger depth did increase the score, especially for relatedness coefficients. The similarity correlation coefficients are almost constant regardless of the weighting function used.

One feature found was that the cosine similarity method for relatedness actually had a much steeper slope upwards compared to the Dice and Jaccard methods. This increased slope was not present for the similarity correlation coefficients.

5.3.3 Effects of Different Depths

Surprisingly enough, differing the maximum allowable depth gave by far the largest variance between the correlation coefficients. When the maximum depth was 1,2 or 3, there were actually not enough intersecting categories in the category hierarchy vectors, and thus a large percentage of article pairs ended up with scores of 0. This threw off the calculation of the correlation coefficients, as it averages the ranks of tied scores. Similarly, going above a depth of 6 with the current algorithm gave too much noise and progressively lowered the correlation coefficient for both WordSim353 and SimSurvey.

A depth of 4 gave the best correlation coefficients, with the coefficient decreasing in steps of between 0.02 and 0.07 depending on the weighting function and vector comparison method for both depths of 5 and 6. WikiRelate! also found that a maximum path length of 4 gave the best results.

5.4 Relatedness Results

Figure 5.1, Figure 5.3 and Figure 5.5 show the relatedness results against the WordSim353 test data set for the Dice, Jaccard and cosine similarity methods respectively. The cosine similarity method had the highest correlation coefficient at 0.5104. This result places CatSim firmly above WikiRelate!, but does significantly worse than ESA and even WLM. The overall results, including the existing methods are detailed in Table 5.1 below.

Table 5.1 Relatedness Correlation Results to WordSim353

Relatedness Measure	Correlation Coefficient
WikiRelate!	0.48
Explicit Semantic Analysis	0.75
Wikipedia Link-based Measure	0.68
CatSim Dice	0.4939
CatSim Jaccard	0.4922
CatSim Cosine Similarity	0.5104

5.5 Similarity Results

Against the SimSurvey test data, CatSim performed much stronger, as expected. The best result was actually the Jaccard method, detailed in Figure 5.4. Figure 5.2 shows the Dice comparison, and Figure 5.6 shows the cosine similarity method results. All three methods are very close together in results, so much so that the category vector comparison method may not matter as much for similarity as it does for relatedness.

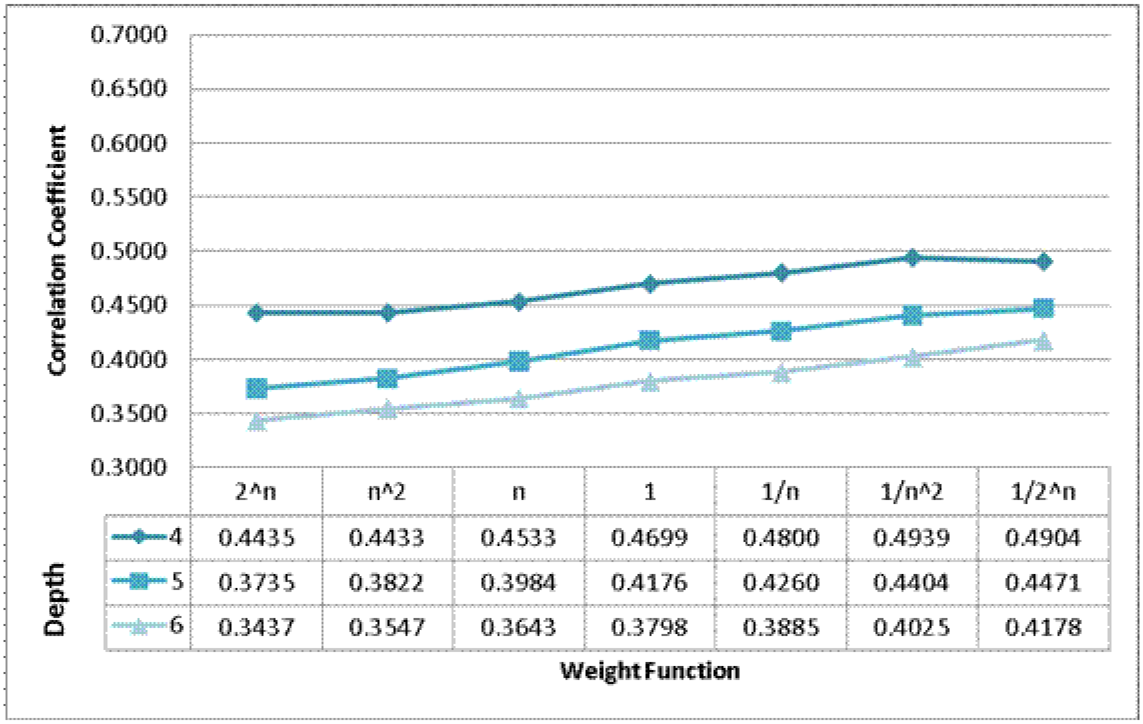
To get a comparison with an existing relatedness method, the WLM measurement was calculated for all of SimSurvey’s named entity pairs using WLM’s open web site[14]. The website accepts two concepts as input and automatically disambiguates those concepts to the Wikipedia articles that are most likely. For our purposes, since we had the exact titles of the articles to compare, the website gave us in all cases the exact articles back, so we were assured of using the same Wikipedia articles for the comparison. Once calculated, WLM is shown to perform 0.09 worse than any of the CatSim methods. Table 5.2 shows the similarity results.

Table 5.2 Similarity Correlation Results to SimSurvey

Similarity Measure	**Correlation Coefficient**
Wikipedia Link-based Measure	0.5921
CatSim Dice	0.6853
CatSim Jaccard	0.6864
CatSim Cosine Similarity	0.6803

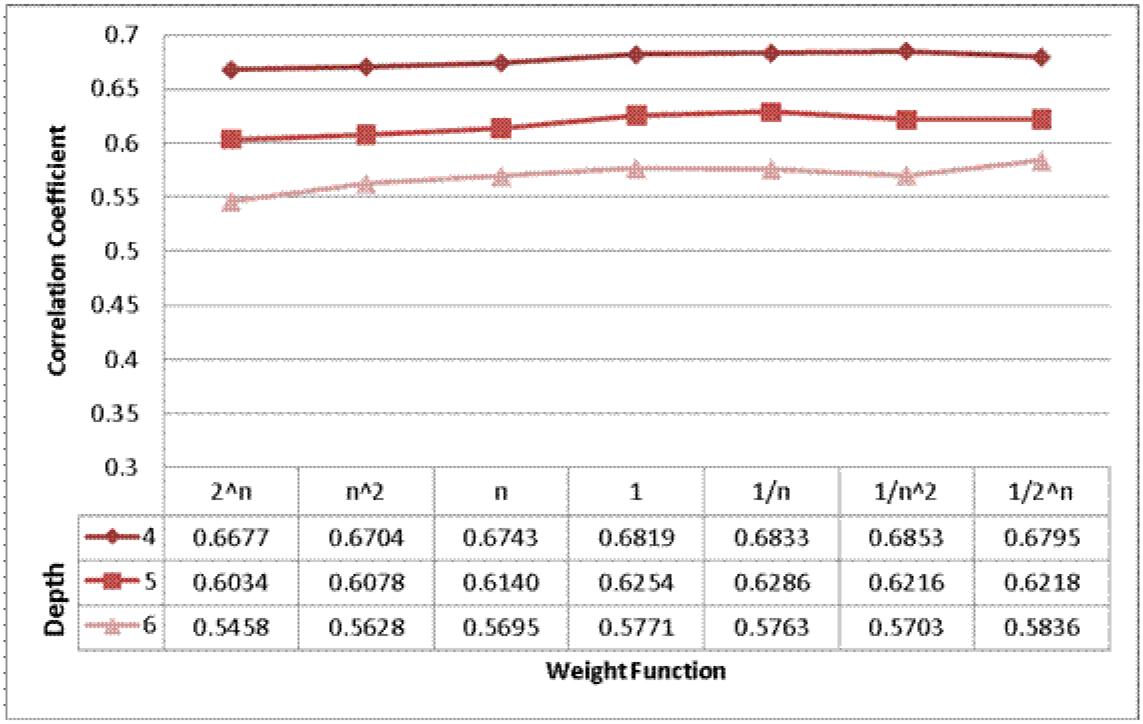
5.6 Overall Impression

Overall, CatSim performed well on the stated task of measuring similarity when compared against human judgments for similarity as given in the SimSurvey results. The comparison methods did not yield as strong a difference as hoped, but the promise of the method is there. Perhaps the most telling point is the difference in the correlation coefficients based on the maximum depth. It shows that arguably the most important thing for CatSim is the construction of the category vector.



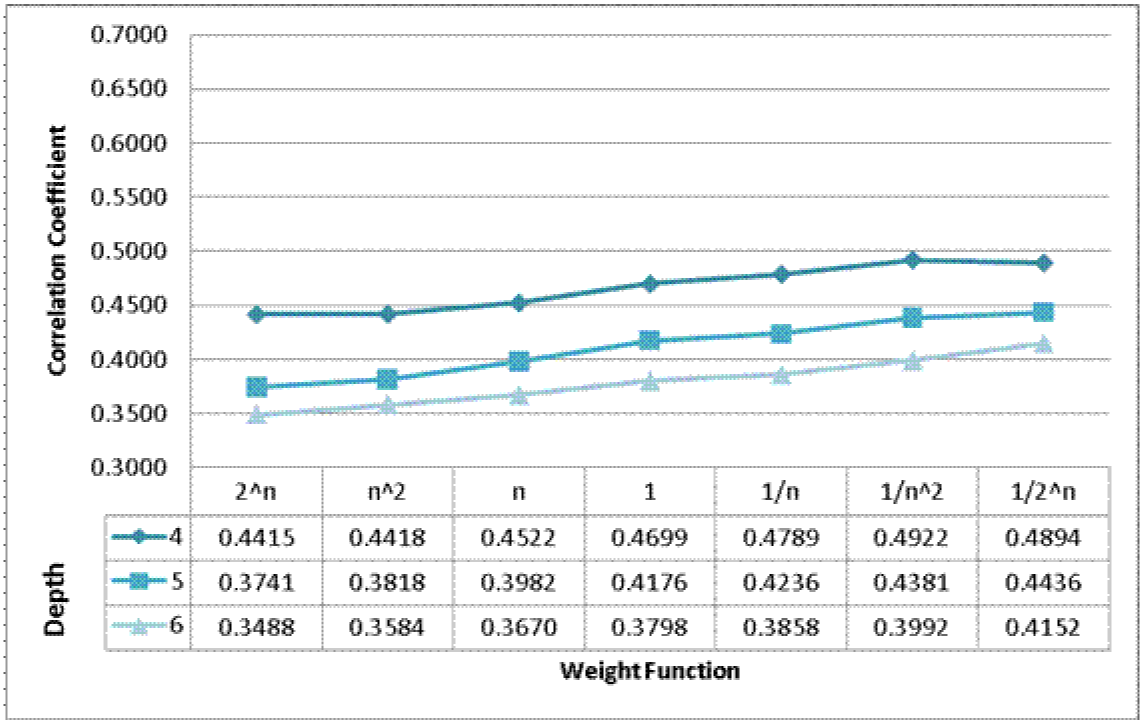
Depth	2^n	n^2	n	1	1/n	1/n^2	1/2^n
4	0.4435	0.4433	0.4533	0.4699	0.4800	0.4939	0.4904
5	0.3735	0.3822	0.3984	0.4176	0.4260	0.4404	0.4471
6	0.3437	0.3547	0.3643	0.3798	0.3885	0.4025	0.4178

Figure 5.1 Dice’s Coefficient: WordSim353



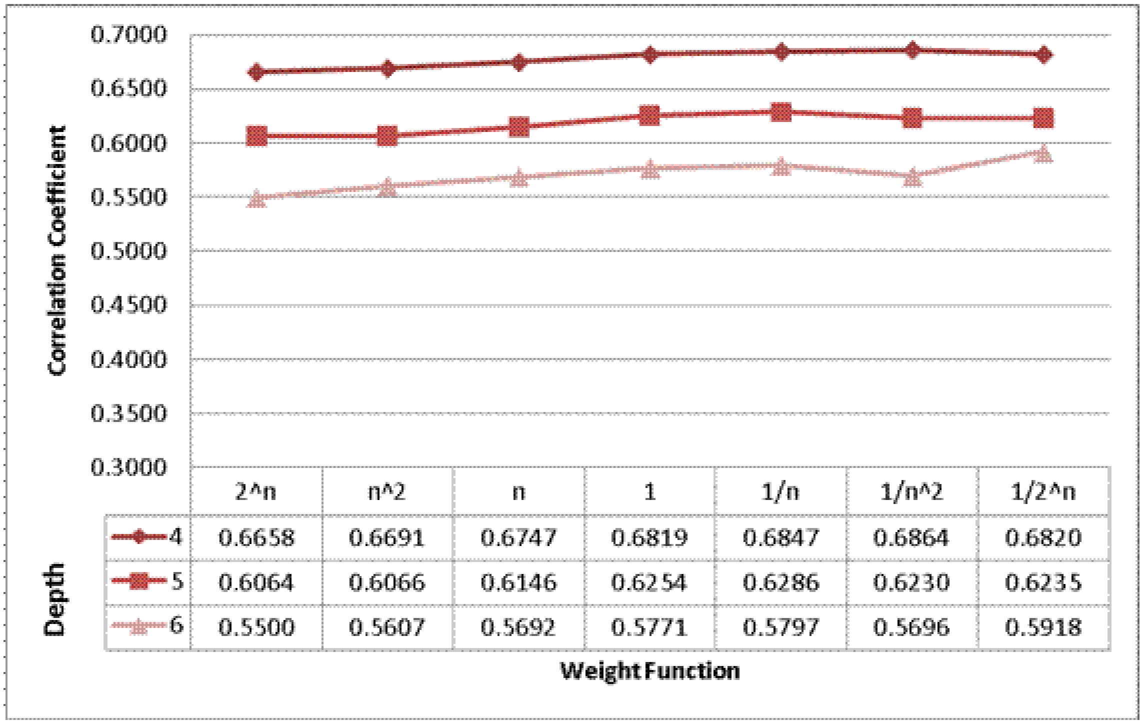
Depth	2^n	n^2	n	1	1/n	1/n^2	1/2^n
4	0.6677	0.6704	0.6743	0.6819	0.6833	0.6853	0.6795
5	0.6034	0.6078	0.6140	0.6254	0.6286	0.6216	0.6218
6	0.5458	0.5628	0.5695	0.5771	0.5763	0.5703	0.5836

Figure 5.2 Dice’s Coefficient: SimSurvey



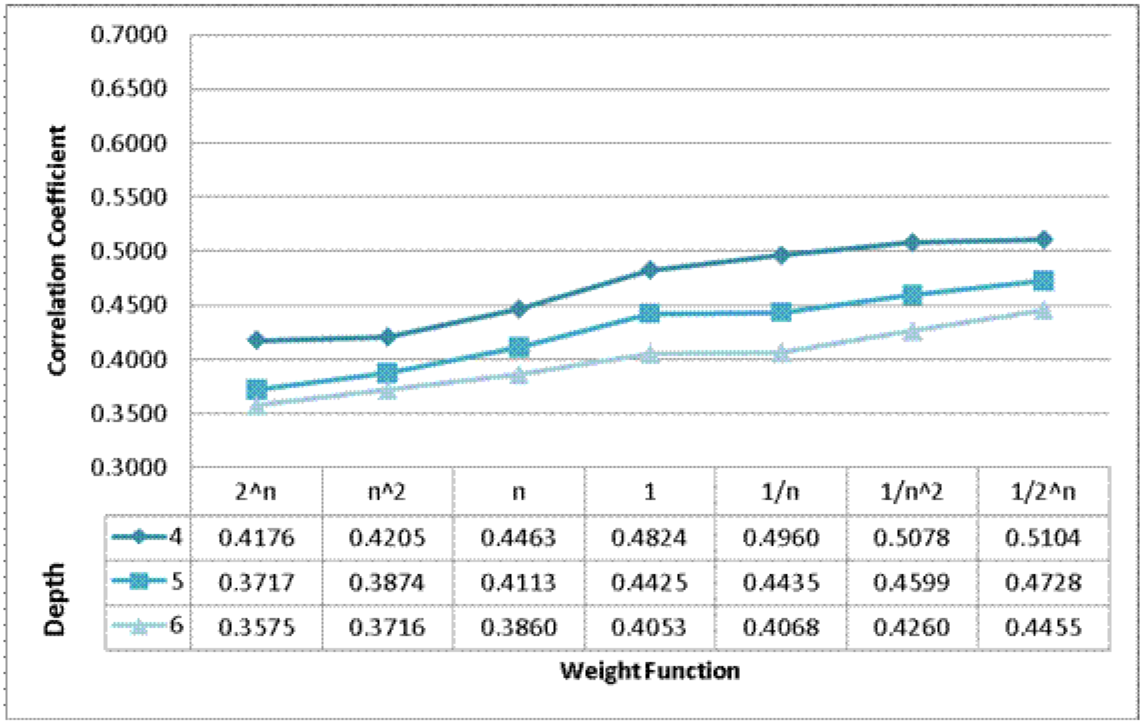
Depth	2^n	n^2	n	1	1/n	1/n^2	1/2^n
◆ 4	0.4415	0.4418	0.4522	0.4699	0.4789	0.4922	0.4894
■ 5	0.3741	0.3818	0.3982	0.4176	0.4236	0.4381	0.4436
▲ 6	0.3488	0.3584	0.3670	0.3798	0.3858	0.3992	0.4152

Figure 5.3 Jaccard Coefficient: WordSim353



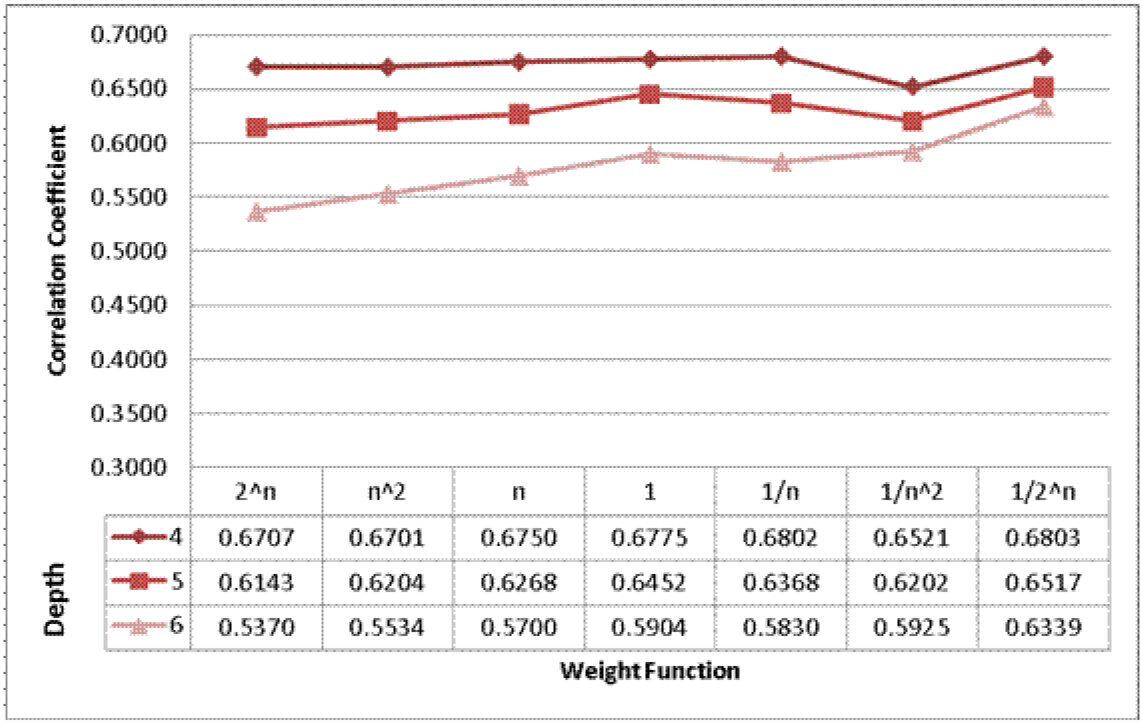
Depth	2^n	n^2	n	1	1/n	1/n^2	1/2^n
◆ 4	0.6658	0.6691	0.6747	0.6819	0.6847	0.6864	0.6820
■ 5	0.6064	0.6066	0.6146	0.6254	0.6286	0.6230	0.6235
▲ 6	0.5500	0.5607	0.5692	0.5771	0.5797	0.5696	0.5918

Figure 5.4 Jaccard Coefficient: SimSurvey



	2^n	n^2	n	1	1/n	1/n^2	1/2^n
◆ 4	0.4176	0.4205	0.4463	0.4824	0.4960	0.5078	0.5104
■ 5	0.3717	0.3874	0.4113	0.4425	0.4435	0.4599	0.4728
▲ 6	0.3575	0.3716	0.3860	0.4053	0.4068	0.4260	0.4455

Figure 5.5 Cosine Similarity: WordSim353



	2^n	n^2	n	1	1/n	1/n^2	1/2^n
◆ 4	0.6707	0.6701	0.6750	0.6775	0.6802	0.6521	0.6803
■ 5	0.6143	0.6204	0.6268	0.6452	0.6368	0.6202	0.6517
▲ 6	0.5370	0.5534	0.5700	0.5904	0.5830	0.5925	0.6339

Figure 5.6 Cosine Similarity: SimSurvey

CHAPTER 6

FUTURE WORK

The work in this thesis creates a framework for extensive future work. The promising results yielded by combining different weighting functions, depth levels and vector comparison methods can be extended to include different weighting functions or better vector comparison methods. The algorithm to perform the category hierarchy collapse can be improved as well. Finally, further improvements in the optimization of the algorithms would allow for improvements in efficiency and processing speed, which could enable the technology to be used in more responsive environments such as web services or embedded within larger NLP applications.

6.1 Improving the Weighting Function

One direction of future work would be to improve which weighting function to use when creating the category vector for an article. As described in Chapter 3, CatSim currently uses a weighting function that is based on the depth that a category in the vector is found on. CatSim could be extended to accept different types of weighting functions that may yield stronger results.

6.1.1 TF-IDF Weighting Function

One option that was considered and discarded due to the additional processing requirements it would impose on the Wikipedia data was to use a TF-IDF based weight of the categories. This type of weighting would more strongly tie the overall use of the category in Wikipedia to the weighted score for it. Combining a TF-IDF weight with a depth based function weight may be even better, as it would use overall use of the category with the specific distance from the article the category was.

6.2 Vector Comparison Methods Revisited

In this thesis, CatSim only uses three different vector comparison methods to compare the articles' category vectors. While Dice, Jaccard, and Cosine Similarity are all common vector comparison methods and ended up with similar correlation coefficients, there exist many others that could give a better correlation coefficient to the test data sets. Some other well known comparison methods that may be worth examining include the overlap coefficient or the Levenshtein distance. Both of these methods can act on weighted vectors to measure the similarity or difference between them.

6.3 Category Collapse Algorithmic Improvements

Perhaps the biggest improvements to CatSim's similarity measures could be found in the way the categories are collapsed. Modifying the algorithm to include or remove categories based on some heuristic may be found to work well. The challenge in dynamically deciding what to include or remove from the category vector lies in what heuristics to use. We considered removing categories based on the usage of the category within Wikipedia, but the preprocessing to enable the removal was outside the scope of this thesis.

Another improvement to the category collapse algorithm could include taking the subcategories into account. Since CatSim only works right now on supercategories, including the subcategories into the algorithm can increase the number of possible common categories between two articles, and yet not increase the maximum path length between a category and the article.

6.4 Efficiency Optimizations

Continuing with the creation of the category vectors, while pulling the data from the database described in Chapter 4 is reasonably quick, the implementation of the category vector creation has room for improvement. Switching from a recursive to an iterative algorithm will yield performance improvements immediately. In addition, implementing a batch processing method to process more than one category and supercategories at a time would yield vast improvements in processing time.

6.5 Extension From Named Entities to Concepts

As described in Chapter 3, CatSim currently works on named entities. A further logical extension of the method would be extend it to using concepts and mapping those concepts to the

Wikipedia articles. Any type of automatic mapping would provide a better method as at that point it could work on text strings and concepts, rather than having to manually map those text strings and concepts. There are some challenges involved with appropriately mapping that made such an automatic mapping out of scope for this thesis, but some ideas for extension are below.

6.5.1 Wikipedia Disambiguation Pages

We attempted to do an automatic mapping between named entities and the Wikipedia articles by making use of Wikipedia’s disambiguation pages. A disambiguation page is created whenever a particular text string may refer to more than one article. An example of this would be for the text string “tiger.” “Tiger” may refer to many different things, including a type of missile, a large feline or any number of military units.

Using these disambiguation pages to decide which article best fits the named entity under consideration as well as the context of the comparison may be possible. We used a naive mapping which indicated that the correct article was the one with the highest similarity measurement to the second named entity. This did not work well in practice, as the context selected sometimes was incorrect. Despite the failure of our mapping method, using the disambiguation pages could provide a robust mapping solution.

6.5.2 Wikipedia Link References

Another possibility for performing automatic mapping while using Wikipedia’s internal structure are the links that point to articles. These links can be used by looking at where they are linking from to determine what the best article to use for the concept would be. An example of this would be to take an article like “Tiger”, and then examine all of the articles that link to “Tiger.” The context determined by those articles may be able to be used to determine the mapping of the comparison concept with respect to the original concept.

<u>6.6 Real World Test Applications</u>

Much of the existing literature runs similarity and relatedness measures against a real world application, like NER or word sense disambiguation to do a “field test” of sorts. As a future work item, a test of CatSim in an actual application based environment to complete a real world task would cement it’s usefulness as a semantic similarity method. The most common test for similarity is word sense disambiguation, and as such it would be an appropriate test for CatSim to complete.

REFERENCES

[1] Bollegala, D., Y. Matsuo, and M. Ishizuka. 2007. Measuring Semantic Similarity between Words Using Web Search Engines. In Proc. of WWW.

[2] Brunstein, A. 2002. BBN's Proposed Answer Categories For Question Answering. <http://www.ldc.upenn.edu/Catalog/docs/LDC2005T33/BBN-Types-Subtypes.html>.

[3] Bunescu, R. and M. Pasca. 2006. Using Encyclopedic Knowledge for Named Entity Disambiguation. In Proc. of EACL, 9-16.

[4] Cucerzan, S. 2007. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In Proc. of Empirical Methods in Natural Language Processing.

[5] Fellbaum, C. (Ed). 1998. WordNet: An Electronic Lexical Database. MIT Press.

[6] Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. 2002. Placing Search in Context: The Concept Revisited, ACM Transactions on Information Systems, 20(1):116-131.

[7] Gabrilovich, E. and S. Markovitch. 2007. Computing semantic relatedness using Wikipedia-based explicit semantic analysis. Proc. of IJCAI, 1606-1611.

[8] Milne, D. and I. Witten. 2008. An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links, In Proc. of AAAI08 Workshop on Wikipedia and Artificial Intelligence.

[9] Miller, G. A. and W. G. Charles. 1991. Contextual correlates of semantic similarity. Language and Cognitive Processes, 6(1):1-28.

[10] Python Software Foundation. 2010. 5 Data Structures - Python v2.7 documentation. <http://docs.python.org/tutorial/datastructures.html#list-comprehensions>.

[11] SQLAlchemy. 2010. SQLAlchemy - The Database Toolkit for Python. <http://www.sqlalchemy.org>.

[12] Strube, M. and S. P. Ponzeto. 2006. WikiRelate! Computing semantic relatedness using Wikipedia. In Proc. of AAAI, 1419-1424.

[13] Wikimedia Foundation. 2010. Wikipedia, the free encyclopedia. <http://en.wikipedia.org/wiki/Main_Page>.

[14] Wikipedia Miner Services. 2010. Wikipedia Miner Services. <http://wdm.cs.waikato.ac.nz:8080/service?task=compare>.

BIOGRAPHICAL INFORMATION

Jared M Ashman graduated December 2010 with his M.S. in Computer Science from the University of Texas at Arlington. Prior to completing his M.S., Jared received his B.S. in Computer Science from Kent State University in December 2005, graduating magna cum laude.

Through his time at the University of Texas at Arlington, Jared's research gravitated towards information retrieval, data mining and natural language processing. His M.S. thesis combined these and dealt with mining Wikipedia's metadata to determine the semantic similarity between named entities.

Professionally, Jared has worked as a software engineer in the healthcare industry, working for the Fortune 500 company Stryker from 2006 until 2008, developing custom imaging software for xray and MRI, and Stryker's electronic medical record management solution. After leaving Stryker, Jared joined the fastest growing privately held company in the United States, Ambit Energy, in 2008 and is currently a development team lead. In his work at Ambit Energy, he leads a team of developers building the software to expand into new energy markets.