5-1-2014

Using the Web 1T 5-Gram Database for Attribute Selection in Formal Concept Analysis to Correct Overstemmed Clusters

Guymon Hall
University of Nevada, Las Vegas, guymon.hall@gmail.com

USING THE WEB 1T 5-GRAM DATABASE FOR ATTRIBUTE SELECTION IN FORMAL CONCEPT ANALYSIS TO CORRECT OVERSTEMMED CLUSTERS

by

Guymon R. Hall

Bachelor of Science (B.Sc.)
University of Arkansas, Fayetteville
2001

A thesis submitted in partial fulfillment of
the requirements for the

Master of Science – Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2014

The Graduate College

We recommend the dissertation prepared under our supervision by

Guymon R. Hall

entitled

Using the Web 1T 5-Gram Database for Attribute Selection in Formal Concept Analysis to Correct Overstemmed Clusters

be accepted in partial fulfillment of the requirements for the degree of

Master of Science – Computer Science

Department of Computer Science

Dr. Kazem Taghva, Ph.D., Committee Chair

Dr. Ajoy Datta, Ph.D., Committee Member

Dr. Matt Pedersen, Ph.D., Committee Member

Dr. Emma Regentova, Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

May 2014

Abstract

Information retrieval is the process of finding information from an unstructured collection of data. The process of information retrieval involves building an index, commonly called an inverted file. As part of the inverted file, information retrieval algorithms often stem words to a common root. Stemming involves reducing a document term to its root. There are many ways to stem a word: affix removal and successor variety are two common categories of stemmers. The Porter Stemming Algorithm is a suffix removal stemmer that operates as a rule-based process on English words. We can think of stemming as a way to cluster related words together according to one common stem. However, sometimes Porter includes words in a cluster that are un-related. This experiment attempts to correct these stemming errors through the use of Formal Concept Analysis (FCA). FCA is the process of formulating formal concepts from a given formal context. A formal context consists of a set of objects, G, a set of attributes, M, and a binary relation I that indicates the attributes possessed by each object. A formal concept is formed by computing the closure of a subset of objects and attributes. Attribute selection is of critical importance in FCA; using the Cranfield document collection, this experiment attempted to view attributes as a function of word-relatedness and crafted a comparison measure between each word in the stemmed cluster using the Google Web 1T 5-gram data set. Using FCA to correct the clusters, the results showed a varying level of success for precision and recall values dependent upon the error threshold allowed.

Acknowledgements

"I would like to thank the members of my advisory committee, particularly Dr. Taghva, for their patience and guidance during my studies. I would especially like to thank my wife, who has supported me unfailingly during these last two years, and has been a constant companion and source of inspiration."

Guymon R. Hall

University of Nevada, Las Vegas
May 2014

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Overview

This thesis is an experiment in the area of information retrieval. Information retrieval deals with the methods and processes of determining meaningful information from a collection of unstructured data. As part of the information retrieval process, information retrieval algorithms often stem words to a common root.

Stemming involves reducing a document term to its root. There are many ways to stem a word: two common categories of stemmers are affix removal stemmers and successor variety stemmers. The Porter Stemming Algorithm is a suffix removal stemmer that operates as a rule-based process on English words [12].

We can think of stemming as a way to cluster related words together according to one common stem. However, sometimes Porter includes words in a cluster that are un-related. This experiment attempts to correct these stemming errors through the use of Formal Concept Analysis.

Formal Concept Analysis is the process of formulating formal concepts from a given formal context. A formal context consists of a set of objects, G, a set of attributes, M, and a binary relation I that indicates the attributes possessed by each object. A formal concept is formed by computing the closure of a subset of objects and attributes, such that the subset of objects contains all objects that possess all of the subset of attributes, and the subset of attributes contains all attributes that possess all of the subset of objects.

Attribute selection is of critical importance in Formal Concept Analysis; using the Cranfield document collection [4], this experiment attempted to view attributes as a function of word-relatedness and crafted a comparison measure between each word in the stemmed cluster using the Google Web 1T 5-gram data set [5].

An *n-gram* is a word-phrase of size n; using a node word as a reference, a collocate is a word that occurs inside the n-gram according to some threshold measure in conjunction with the reference node. This experiment formed a comparison measure that utilized the list of collocates

contained in the Google Web 1T 5-gram data for each word in the cluster ranked by frequency of occurrence; it then calculated a comparison value for each word-pair in the cluster using a modified Dice comparison technique [5]. Using Formal Concept Analysis to correct the clusters, the results showed a varying level of success for precision and recall values dependent upon the error threshold allowed. The process was successful in correcting stem clusters that contained un-related words; however, the process also induced error into stem clusters that did not contain un-related words.

This thesis is composed of five chapters. The first chapter is the introduction you just read. Chapter two is an introduction to information retrieval, and it covers tokenization and stop word removal. Chapter three focuses on stemming and the Porter stemmer; successor variety stemmers and affix removal stemmers are introduced, and then a brief analysis of the Porter stemmer is given. The fourth chapter is an introduction to Formal Concept Analysis. Formal contexts and formal concepts are defined, an overview of the In-Close Algorithm is provided, and a discussion of concept lattices follows. The fifth chapter provides the details of the experiment and its results; experiment methodology, data sources, and precision and recall definition and values are examined. The sixth chapter covers conclusions and future recommendations.

Chapter 2

Information Retrieval

Information retrieval is a burgeoning field of academic study with very practical implications for everyday life. Formally speaking, *information retrieval* is defined as "finding material of an unstructured nature that satisfies an information need from within large collections [11]."

In this context, the term *unstructured data* refers to data which does not have a formally defined organization applied to it. For example, a data set that does not conform to the principles underlying relational databases could be unstructured data. What constitutes a large collection also bears consideration. Document collections ranging from millions to billions of individual documents are not uncommon. The largest unstructured document collection is the entire World Wide Web comprised of all publicly accessible web pages.

With large document collections of unstructured data, a variety of methods and algorithms are needed to be able to efficiently sift through all the data and retrieve the desired information. These techniques need to be able to process large amounts of information quickly, allow for flexibility in matching operations, and provide some type of either ranked or categorized results.

The first step to facilitate these objectives is to build an inverted file. An inverted file is a dictionary of terms that for each term, contains a list of documents in which the term occurs [11]. The process of building an inverted file includes tokenization, removal of stop words, and stemming.

2.1 Tokenization

Tokenization consists of fragmenting a character sequence, usually an entire document, into its individual tokens. A token is "a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing [11]."

The easiest way to tokenize a document collection is to simply split the document according to whitespace characters. This turns out to be a fairly trivial way of tokenizing a document collection;

tokenizing according to whitespace can yield inaccurate results and lead to data being included in the information retrieval process that is not relevant.

2.1.1 Language Identification

There are a number of considerations to take into account when tokenizing a document. The first is to determine the language of the document collection. This can be manually given, or an automated language identifier can be used. Issues in the tokenization of a document collection tend to be language specific, so determining the correct language of a document is critical in applying any remaining tokenization schemes to the document [11].

2.1.2 Hyphenation

The second issue is to determine what constitutes a correct token. Hyphenation is generally used in a variety of instances depending on the particular language of the document. In the English language, hyphenation is used in instances ranging from delineating vowel prefixes in words such as *co-education* to joining multiple words together to form complex compound word phrases. Handling hyphenation in a document can be solved by using classification means or by using rule-based methods [11].

2.1.3 Punctuation

The way a language is punctuated also affects how a document in that language should be tokenized. For example, the use of apostrophes in the English language presents some difficulty in handling the tokenization process: a word such as *aren't* should remain as a single token, but a word such as *Harry's* could be tokenized as *Harry*. Other languages might have a lack of familiar punctuation rules that would make it difficult to parse a document collection into tokens. The use of machine learning models contribute greatly to the ability to properly account for punctuation of a document in a particular language.

2.1.4 Compound Words

Compound words form another consideration to deal with in the tokenization process. Highly language dependent, compound words can be identified by means ranging from hyphenation patterns to a simple concatenation of individual words together to form a single compound word. Properly tokenizing a compound word can be accomplished using a compound-splitter module, which examines a compound word for sub-words that occur within the language [11].

2.2 Stop Word Removal

The next step in building an inverted file involves removal of stop words. *Stop words* are high-frequency terms that occur as a large percentage in common use of a particular language that most likely do not contribute semantic meaning to the document. Examples of stop words include *a*, *and*, *the*, etc. Information retrieval systems have tended in recent years to refrain from removing stop words; text units such as song titles and stanzas of poems could be adversely affected by removing stop words from the indexed collection [11]. There are two main ways in which to remove stop words.

The first method is to build a list of stop words that is unique to the document. The method involves sorting the terms in a document collection according to collection frequency, and then to take the most frequent terms as representing a list of stop words unique to that document. Considerations such as Zipf's Distribution must be taken into account [11].

The second method is to utilize a pre-built list of stop words as determined by general research into the particular language. This stop list would be less specific to the actual document collection, but could be more representative of actual stop words in that language.

Chapter 3

Stemming

The next step in building an inverted file is to stem the remaining terms of the document collection. A stemming algorithm is an algorithm "which reduces all words with the same root...to a common form [10]." Research has shown that some form of stemming as part of the information retrieval process yields an improvement in the resulting data [9]. Two common types of stemming algorithms include successor variety algorithms and affix removal algorithms.

3.1 Successor Variety

Successor variety stemming is a stemming algorithm that attempts to find the stem of a word by taking into consideration the morphological variants of the word and analyzing the prefixes to determine the longest common prefix. The way in which the longest common prefix is determined varies by each algorithm, but it can be thought of in terms of graph theory through construction of a suffix tree [15].

Given such a construct, determining a stem consists of constructing a reasonable path from the root to a subtree that accounts for all words in question. Nodes that have high outdegree become candidates for a common stem [15].

3.2 Affix Removal

Affix removal stemming algorithms function by removing suffixes, prefixes, or both from words to produce a stem. Often, the stem closely approximates the grammatical root of the word [6]. Most of the time, affix removal stemming algorithms are suffix-strippers: they strip each word of a suffix by following a pre-determined series of steps and rules.

Lovins describes two approaches to constructing affix removal stemming algorithms. The *iteration* principle is based on the idea that suffixes are attached to stems following a certain order

or method. That is, there are some suffixes that are attached before other suffixes; removing the suffixes in this case involves iteratively reversing the order of suffix addition [10].

The *longest-match* principle involves finding the longest-matching affix of a word given a set of possible affixes. Lovins states, "All possible combinations of affixes are compiled and then ordered on length. If a match is not found on longer endings, shorter ones are scanned [10]."

3.3 The Porter Stemming Algorithm

The Porter Stemming Algorithm is an affix removal stemmer that functions by removing suffixes according to a list of rules representing suffix rules of the English language. Porter's algorithm has been shown to have several benefits during the information retrieval process. Porter showed that his algorithm reduced the size of the vocabulary in a given document collection by about one-third [12]; for the purposes of this experiment, his method is accurate in clustering related words together about 99.998% of the time. Porter's algorithm consists of a series of five steps applied sequentially, each step of which filters the word through a series of grammar rules.

Porter defines a *consonant* as a letter other than A, E, I, O, or U, and not including Y when Y is preceded by a consonant. He defines a *vowel*, v, as a letter that is not a consonant. He defines the *measure*, m, as the length of any word or word part [12]. He uses these definitions to formulate his rules in each of the steps:

Algorithm 1: Porter Stemming Algorithm

```
Step 1a
    [1] SSES → SS  // caresses → caress
    [2] IES → I  // ponies → poni, ties → ti
    [3] SS → SS  // caress → caress
    [4] S →  // cats → cat
Step 1b
    if m > 0 then
        [1] EED → EE  // feed → feed, agreed → agree
    end
    if *v* then
        [2] ED →  // plastered → plaster, bled → bled
        [3] ING →  // motoring → motor, sing → sing
    end
    if Rule 2 or 3 is successful then
        [6] AT → ATE  // conflat(ed) → conflate
        [7] BL → BLE  // troubl(ed) → trouble
        [8] IZ → IZE  // siz(ed) → size
        if *d and not (*L or *S or *Z) then [4] → single letter  // tann(ed) → tan

        if (m = 1 and *o) then [5] → E  // fail(ing) → fail, fil(ing) → file

    end
```

Step 1c

if $*v*$ **then**

[1] → I // *happy → happi, sky → sky*

end

Step 2

if $m > 0$ **then**

[1] ATIONAL → ATE // *relational → relate*

[2] TIONAL → TION // *conditional → condition,*

[3] ENCI → ENCE // *valenci → valence*

[4] ANCI → ANCE // *hesitanci → hesitance*

[5] IZER → IZE // *digitizer → digitize*

[6] ABLI → ABLE // *conformabli → conformable*

[7] ALLI → AL // *radicalli → radical*

[8] ENTLI → ENT // *differentli → different*

[9] ELI → E // *vileli → vile*

[10] OUSLI → OUS // *analogousli → analogous*

[11] IZATION → IZE // *vietnamization → vietnamize*

[12] ATION → ATE // *predication → predicate*

[13] ATOR → ATE // *operator → operate*

[14] ALISM → AL // *feudalism → feudal*

[15] IVENESS → IVE // *decisiveness → decisive*

[16] FULNESS → FUL // *hopefulness → hopeful*

[17] OUSNESS → OUS // *callousness → callous*

[18] ALITI → AL // *formaliti → formal*

[19] IVITI → IVE // *sensitiviti → sensitive*

[20] BILITI → BLE // *sensibiliti → sensible*

end

Step 3

if $m > 0$ **then**

[1] ICATE → IC // *triplicate → triplic*

[2] ATIVE → // *formative → form*

[3] ALIZE → AL // *formalize → formal*

[4] ICITI → IC // *electriciti → electric*

[5] ICAL → IC // *electrical → electric*

[6] FUL → // *hopeful → hope*

[7] NESS → // *goodness → good*

end

Step 4

if $m > 1$ **then**

[1] AL → // *revival → reviv*

[2] ANCE → // *allowance → allow*

[3] ENCE → // *inference → infer*

[4] ER → // *airliner → airlin*

[5] IC → // *gyroscopic → gyroscop*

[6] ABLE → // *adjustable → adjust*

[7] IBLE → // *defensible → defens*

[8] ANT → // *irritant → irrit*

[9] EMENT → // *replacement → replac*

[10] MENT → // *adjustment → adjust*

[11] ENT → // *dependent → depend*

if $*S$ *or* $*T$ **then** [12] ION → // *adoption → adopt*

end

```
Step 4 cont'd
   if m > 1 then
      [13] OU → // homologou → homolog
      [14] ISM → // communism → commun
      [15] ATE → // activate → activ
      [16] ITI → // angulariti → angular
      [17] OUS → // homologous → homolog
      [18] IVE → // effective → effect
      [19] IZE → // bowdlerize → bowdler
   end
Step 5a
   if m > 1 then
      E → // probate → probat, rate → rate
   end
   if m = 1 and not *o then
      E → // cease → ceas
   end
Step 5b
   if m > 1 and *d and *L then
      → single letter // controll → control, roll → roll
   end
```

It should be apparent that automated stemming algorithms do not always yield the exact grammatical root of a word, nor are they intended to do so. Another observation is that we can view words that Porter reduces to the same stem as forming a cluster. An example of this can be seen using the words *include*, *includes*, and *including*. Each of these words are stemmed by Porter to the stem *includ*. We write this occurrence as:

$$\text{includ} \Rightarrow \text{include, includes, including}$$

Thus, the three words *include*, *includes*, and *including* can be said to form part of a cluster that is identified by the stem *includ*.

One issue that arises when viewing stemming in this way is how to deal with situations in which grammatically unrelated words are given the same stem. An example of this can be seen as follows:

$$\text{experi} \Rightarrow \text{experiment, experiments, experience, experiences}$$

In the above cluster, the words *experiment* and *experiments* are really unrelated to the words *experience* and *experiences*, yet they each have the same stem *experi*. This thesis attempts to address that situation through the use of Formal Concept Analysis.

Chapter 4

Formal Concept Analysis

Formal Concept Analysis is a method for constructing formal concepts using operations on a formal context. Formal Concept Analysis was first written about by Rudolf Wille in the early 1980s [17]. Formal Concept Analysis finds application in a wide range of disciplines: linguistics, artificial intelligence, and information retrieval are but a few of the disciplines that make use of formal concept analysis [13].

4.1 The Mathematics of Formal Concepts

A *formal context* is a triple, (G, M, I), where G is a set of objects, M is a set of attributes, and $I \subseteq G \times M$ is a binary relation such that gIm indicates that object $g \in G$ possesses attribute $m \in M$ [17]. We can represent a formal context as a table consisting of n objects and m attributes such that:

$$a_{ij} = \begin{cases} 1 & \text{if } g_i I m_j \text{ is true} : 0 \leq i \leq n,\ 0 \leq j \leq m \\ 0 & \text{otherwise} \end{cases}$$

e.g.,

	m_0	m_1	...	m_m
g_0	1	1		
g_1	0	1		
$\vdots$				
g_n	1			1

Table 4.1: A formal context

Let

$$A' = \{m \in M : gIm \ \forall \ g \in A : A \subseteq G\}, \text{ and}$$
$$B' = \{g \in G : gIm \ \forall \ m \in B : B \subseteq M\}$$

A *formal concept* is a pair, (A, B), such that $A = B'$ and $B = A'$. The pair (A, B) form a relation which is closed. We say that A is the *extent* of the formal concept, and B is the *intent* of the formal concept [17].

A simple illustration using formal concepts can be made with the following example:

	m_0	m_1	m_2
g_0	1	1	0
g_1	0	1	0
g_2	1	0	1

Table 4.2: Formal concept example

A simple scan of the table shows that there are four formal concepts:

$$c_0 = \{g_0, g_2\}, \{m_0\}$$
$$c_1 = \{g_0\}, \{m_0, m_1\}$$
$$c_2 = \{g_2\}, \{m_0, m_2\}$$
$$c_3 = \{g_0, g_1\}, \{m_1\}$$

4.2 The In-Close Algorithm

Many algorithms have been developed to compute the closure of sets to determine all formal concepts within a given formal context. Ganter's algorithm for computing formal concepts was published in 1984 [7]. The In-Close algorithm, developed by Andrews in 2009, is a recursive algorithm that uses incremental closure operations to compute all formal concepts in a formal context [2].

In-Close is initialized with the set of all objects, and an empty set of attributes. Then, for each of the attributes, if an object possesses that attribute it is added to the new formal concept. Then, for the set of objects in the new formal concept, the algorithm recursively computes the closure of that concept by passing the new set of objects to the next function call. This process repeats until all attributes are exhausted [2].

Given a formal context with n objects and m attributes, the algorithm runs thusly:

Algorithm 2: In-Close Algorithm

```
for attributes j = y to m_m do
    initialize a new formal concept, R_1;
    foreach object i ∈ R_0 do
        if i has attribute j then
            insert i into R_1's extent;
        end
    end
    if R_1.size > 0 then
        if R_1.size == R_0.size then
            insert j into R_0's intent;
        end
        else
            if R_1 is canonical then
                insert j into R_1's intent;
                InClose(R_1, j + 1)
            end
        end
    end
end
```

The algorithm hinges on the fact that it computes formal concepts according the lexicographical, or canonical, ordering of the intent of each concept. Thus, for each newly formed extent the algorithm checks to see if it has already been formed, and if it has, the intent is merely updated for the already existing extent and no recursion is necessary.

In experimental testing, the algorithm has been shown to run more efficiently than similar algorithms [2]. It can be seen through analysis of the algorithm that In-Close runs with an upper-bound complexity of $O(m^2n + m^3)$ for a context with n objects and m attributes.

4.3 Example

In order to illustrate the operation of the In-Close Algorithm, we can use the following formal context as an example:

	furry	tail	legs
cat	1	1	1
dog	1	1	1
turtle	0	1	1
fish	0	1	0

Table 4.3: A selection of animals and their attributes

Given this formal context, we initialize In-Close with $R_0 = \{cat, dog, turtle, fish\}$ and $y = 0$:

R	y	extent	intent	canonical ?
R_1	$y = 1$	{cat, dog}	{furry}	yes
R_1	$y = 2$	{cat, dog}	{furry, tail}	no
R_1	$y = 3$	{cat, dog}	{furry, tail, legs}	no
R_2	$y = 2$	{cat, dog, turtle, fish}	{tail}	yes
R_3	$y = 3$	{cat, dog, turtle}	{tail, legs}	yes
R_4	$y = 3$	{cat, dog, turtle}	{tail, legs}	no

Table 4.4: Computation of the In-Close algorithm

Thus, for this example, there are three formal concepts:

$$R_1 = \{\text{cat, dog}\}, \{\text{furry, tail, legs}\}$$
$$R_2 = \{\text{cat, dog, turtle, fish}\}, \{\text{tail}\}$$
$$R_3 = \{\text{cat, dog, turtle}\}, \{\text{tail, legs}\}$$

The concepts are computed in order of their canonical appearance according the operation of the algorithm.

Only those intents that are not previously found via a canonical search constitute a new formal concept. For instance, R_1's extent is initially computed as {cat, dog}, its intent is {furry}, and it is initially canonical; however, because both *cat* and *dog* have the attributes *furry* and *tail*, each recursive iteration on the next attribute shows that the newly computed intent is not canonical: thus, R_1's intent is simply added to instead of generating a new formal concept.

Thus, in this example, R_4 is not included as a formal concept because it is not canonical.

4.4 Concept Lattices

Another way in which Formal Concept Analysis can be visualized is through the construction of a *concept lattice*. A concept lattice consists of a set of formal concepts and the subconcept-superconcept relation between the concepts [13]. We can use the example above to illustrate a concept lattice (Figure 4.1).

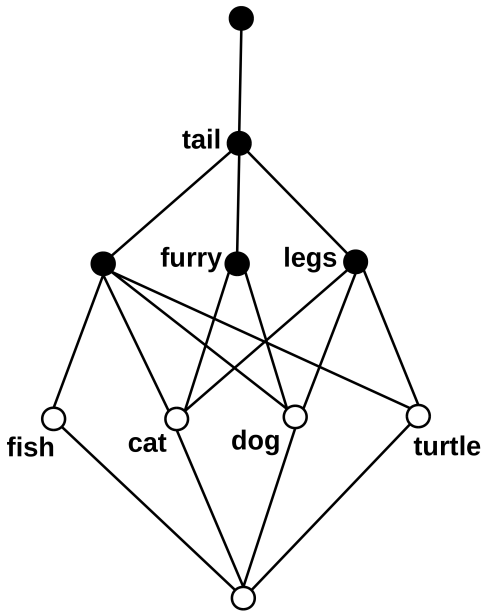

Figure 4.1: Concept Lattice

For a given concept lattice, to determine a formal concept, one needs to find the extent of a formal concept by tracing all paths which lead "down" from the node to collect all the objects and the intent by tracing all paths which lead "up" to collect all the attributes [13].

Given two formal concepts, we can define this subconcept-superconcept relation as follows:

$$(A_1, B_1) \leq (A_2, B_2) \iff A_1 \subseteq A_2 \text{ and } B_1 \supseteq B_2 \quad [17]$$

Thus, for two formal concepts in the example, say

$$R_1 = \{\text{cat, dog}\}, \{\text{furry, tail, legs}\}$$
$$R_2 = \{\text{cat, dog, turtle, fish}\}, \{\text{tail}\}$$

We can say that $R_1 \leq R_2$, since

$$\{\text{cat, dog}\} \subseteq \{\text{cat, dog, turtle, fish}\}, \text{ and}$$
$$\{\text{furry, tail, legs}\} \supseteq \{\text{tail}\},$$

and we can describe this in terms of the concept lattice as follows:

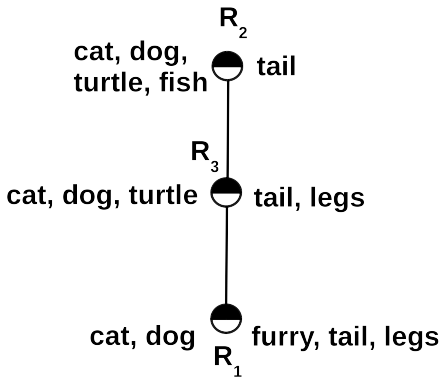

Figure 4.2: Concept Hierarchy

4.5 Implications

Table 4.3, in conjunction with Figure 4.1, show that certain dependency relationships exists between some of the attributes in the formal context. For example, one can see that every object that possesses the attribute *furry* also possess the attributes *tail* and *legs*. Similarly, every object that possesses *legs* as an attribute also has the attribute *tail*.

We can write these dependencies as

$$\{\text{furry}\} \rightarrow \{\text{tail, legs}\}$$
$$\{\text{legs}\} \rightarrow \{\text{tail}\}$$

Identifying these dependencies allows the ability to gain efficiency in exploring formal concepts for knowledge generation. If we know that every object that possesses attribute a_1 also possesses attribute a_2, then we don't need to use computation time to calculate formal concepts with attribute a_2 [16].

4.6 Association Rules

Figure 4.2 shows the hierarchical relationship between formal concepts R_1, R_2, and R_3. The fact that such a relationship exists and can be articulated from the concept lattice has further application, particularly for formal contexts where large quantities of formal concepts might be generated.

Attribute exploration benefits greatly from the existence of these relationships. In conjunction with implications, attribute exploration can be used to prune the lattice using a greatest common

subconcept methodology [16]. For example, if we were to view Figure 4.2 as a sublattice of some larger concept lattice, if the information to be retrieved does not contain the attribute tail, then we can prune the lattice at R_2, including all subconcepts, and therefore save the computational time required to search through those concepts.

Attribute exploration has been shown to have application in many areas. Aranda-Corral, Diaz, and Galan-Paez showed that using attribute exploration in sports forecasting resulted in a significant increase in predictive accuracy [3].

Chapter 5

Applying Formal Concept Analysis to Stem Clusters

This experiment viewed stemming as a way to cluster related words together. For those clusters in which un-related words are grouped together, it attempted to use Formal Concept Analysis to refine those un-related terms into distinct clusters.

5.1 Data Sources

This experiment was conducted using the Cranfield document collection. The Cranfield document collection is a set of 1400 documents that was first used as part of the Cranfield experiments in the 1960s [4]. The Cranfield experiments are widely held to mark the beginning of the modern era of automated information retrieval systems [8].

The documents in the Cranfield document collection are largely of a technical nature. Whether or not the nature of the documents, being highly specialized, had any bearing on experimental results will be discussed in the section on experimental conclusions.

5.2 Building Clusters

This experiment followed the general flow of information retrieval systems. First, the document was tokenized. Then, stop words were removed. After stemming each remaining term, clusters were then formed.

5.2.1 Tokenization

The first step of the experiment was to tokenize the document collection. This was done in a fairly trivial way. Tokens were created using whitespace as a delimiter.

5.2.2 Stop Word Removal

The next step was the removal of stop words. A pre-defined list of common English stop words was used.

5.2.3 Stemming

After stop word removal, each remaining term was stemmed using the Porter stemmer.

5.2.4 Clustering

During the stemming process, the original word was retained prior to being stemmed. Then clusters were formed for each stem containing the words that had that stem in common. Clusters that contained only one term were not retained. Thus, there were a total of 5300 clusters considered in this experiment.

5.3 Building Formal Contexts

After parsing the document collection into clusters, the next step was to form formal contexts for each cluster. In each formal context, the objects were defined as each term in the cluster. Attribute selection was given as a function of word-relatedness for each object-pair in the context. Then, for each context, the mean, standard deviation, and range for the comparison values was calculated.

5.3.1 Attribute Selection

In determining the attributes for each object in the context, a function, N, was defined that calculated a comparison value for each word pair. This resulted in an $n \times n$ table:

	$N(w_i, w_0)$	$N(w_i, w_1)$	...	$N(w_i, w_n)$
w_0	1.0	0.27		0.19
w_1	0.27	1.0		
$\vdots$				
w_n	0.19			1.0

Table 5.1: Formal context example for a stem cluster

5.3.2 Comparison Value

The function, N, that calculated the comparison value for each object-pair in the context was based on a modified-Dice comparison of collocates. A *collocate* is a "recurrent and predictable word combination, which [is] a directly observable property of natural language [5]." Collocates then are words which commonly occur together with a node word in n-grams.

The collocate data for the experiment was taken from the Google Web 1T N-gram data set, which contains English n-grams and their observed frequency statistics. For each object in the context, a list of the top 1000 collocates, ranked by frequency occurrence was retrieved. Then, for each object-pair, the lists of collocates were compared using a modified-Dice measure of similarity:

$$\frac{2C}{|A| + |B|}$$

where C is the number of collocates in common, and A and B are the list of collocates for each object.

The key thought for this experiment is that comparisons between words which are related should have higher comparison values than do comparisons between un-related words. This is derived from Firth's definition of *collocate*: "You shall know a word by the company it keeps [5]."

5.4 Conducting the Experiment

The set of formal contexts were randomly divided into 60% training and 40% testing. In the training phase, 500 random word comparisons were calculated using the previously methodology and the mean and standard deviation of these random values was calculated. Two words selected randomly from the document have a low probability of being related; thus, comparison values that fall within three standard deviations of the random mean are most likely un-related.

For each of the formal contexts, both correct and over-clustered, in the training set, the random comparison mean, μ_r and standard deviation, σ_r, was used to calculate a z-score for each comparison value x in the context. The z-score was calculated as follows:

$$z = \frac{x - \mu_r}{\sigma_r}$$

This method of using the z-score as an estimate of probability is similar to the method used by Acerbi, Lampos, Garnett, and Bentley in their research into books of the 20th century [1].

Then, the mean, standard deviation, and range of values was calculated for the entire context and stored. These statistics formed the basis for use in applying the experiment to the testing data.

After these statistics were calculated, they were then applied to the testing data. For each of the formal contexts in the test data, if the formal context's range was greater than four standard deviations from the average range of correct clusters, as determined in the training data, the values of the context were normalized using that context's mean and standard deviation. The resulting comparison values yielded a formal context in which positive values existed for related word comparisons, and negative values for un-related word comparisons.

5.5 Example

To illustrate the process, an example will show how the results were obtained. Consider the following formal context:

experi	experiment	experiments	experience	experiences
experiment	1.0	0.54	0.39	0.36
experiments	0.54	1.0	0.34	0.34
experience	0.39	0.34	1.0	0.59
experiences	0.36	0.34	0.59	1.0

Table 5.2: Example formal context from Cranfield collection

The average for 500 random comparisons was 0.135 and the standard deviation for that average was 0.098. Given these values, each comparison in the formal context was normalized to its z-score using the random average and standard deviation:

experi	experiment	experiments	experience	experiences
experiment	8.83	4.15	2.62	2.26
experiments	4.15	8.83	2.13	2.07
experience	2.62	2.13	8.83	4.68
experiences	2.26	2.07	4.68	8.83

Table 5.3: The formal context after adjusting according to random comparisons

Training data showed an average range for correct formal contexts of 0.186 with a standard deviation of 0.99. When we compare each formal context in the testing data using these values, we

see that our example formal context has a range of 6.76, which is greater than 4 standard deviations of the training range mean. Thus, we normalize the formal context, yielding the following values:

experi	experiment	experiments	experience	experiences
experiment	2.22	0.15	-0.53	-0.69
experiments	0.15	2.22	-0.75	-0.77
experience	-0.53	-0.75	2.22	0.38
experiences	-0.69	-0.77	0.38	2.22

Table 5.4: The formal context after normalization

When the In-Close algorithm is applied, we consider non-negative values to be indicative of an object possessing that attribute, and negative values indicate that an object does not possess that attribute. Thus, the formal context becomes:

experi	experiment	experiments	experience	experiences
experiment	1	1	0	0
experiments	1	1	0	0
experience	0	0	1	1
experiences	0	0	1	1

Table 5.5: The formal concept in terms of its binary attributes

The In-Close algorithm computes the formal concepts for the example formal context as follows. We initialize the algorithm with

$$R_0 = \{\text{experiment, experiments, experience, experiences}\},$$

and

$$y = 0\text{:}$$

R	y	extent	intent	canonical ?
R_1	$y = 1$	{experiment, experiments}	{experiment}	yes
R_1	$y = 2$	{experiment, experiments}	{experiment, experiments}	no
R_2	$y = 2$	{experiment, experiments}	{experiments}	no
R_3	$y = 3$	{experience, experiences}	{experience}	yes
R_3	$y = 4$	{experience, experiences}	{experience, experiences}	no

Table 5.6: Steps of the In-Close algorithm applied to the formal context

Thus, for this example, the process results in two formal concepts, with the extent of each concept listed as follows:

$$R_1 = \{\text{experiment, experiments}\}$$
$$R_3 = \{\text{experience, experiences}\}$$

It is easy to see that each extent represents a corrected refinement of the original cluster that the Porter Stemming Algorithm yielded. Thus, for this formal context, the process has correctly separated out un-related words from the original cluster, improving the accuracy of the clustering process.

5.6 Results

The results for the experiment show a varying level of success based on an allowed error threshold. Results were calculated using precision and recall values as discussed by Reynaert [14]. Each formal context could have four possible results: a correct context could remain correct; this is a true negative (TN). A correct context could have erroneously been changed during the process. This is a false positive (FP). An over-clustered context could have been correctly adjusted; this is a true positive (TP). Finally, an over-clustered context could still be incorrect after the process, yielding a false negative (FN).

Precision and recall were calculated, then, as follows:

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

Given an error threshold as a percentage of induced errors, the results were as follows:

error threshold	recall	precision
0.00% (0 / 576)	0.00	1.00
0.53% (3 / 567)	0.20	0.40
1.03% (6 / 582)	0.17	0.40
6.22% (35 / 563)	0.14	0.03
8.85% (51 / 576)	0.50	0.06
26.90% (156 / 580)	0.57	0.03

Table 5.7: Precision and recall results for given error thresholds

Thus, when no error was allowed, the process failed to correct any of the over-clustered contexts. When only 0.5% error was allowed, the process corrected a few of the over-clustered contexts, yielding a recall value of 0.2, but it also induced 3 additional errors in correct formal contexts, reducing the precision value to 0.4.

Chapter 6

Conclusions & Future Considerations

The results of the data are mixed. On the one hand, the process clearly did a good job of correcting those formal contexts that contained unrelated words as evidenced by the recall results. On the other hand, the process also induced errors in already correct formal contexts. Notice that the more formal contexts we attempted to correct, the more errors were induced in correct contexts.

The key issue hinged on the ability to distinguish between correct clusters and clusters that needed refinement. The normalization process and resulting computation of formal concepts generally produced good results when it was applied to clusters that were in fact in need of such refinement. However, this experiment was unable to find a satisfactory method to distinguish between correct clusters that should not have had this process applied to them, and incorrect clusters that contained un-related word pairs. When the normalization process was applied to correct clusters, the results show the errors that were induced and impacted the precision of the experiment.

6.1 Future Work

This experiment, while not producing optimal results, gave rise to a number of interesting questions that could be options for further research in this area.

6.1.1 Variety Across the Document Collection

The first option is to discern how much the document collection influenced the results. In other words, did the fact that the Cranfield collection is a rather homogeneous collection of engineering documents skew the results? Thus, this experiment should also be reproduced using other

document collections as well, such as the Brown Corpus.

6.1.2 Word-Comparison Calculation

The second area is to research other ways to compute word-comparison values. While using the modified-Dice coefficient with collocates showed promise in most of the formal contexts, the distinction was not such that it was useful as an attribute for formal contexts.

6.1.3 Attribute Selection

Related to that notion is the need to research other attributes for the formal contexts. Perhaps using other properties of words, such as part of speech, language derivation, etc., would yield other attributes that would provide a more solid footing for correctly distinguishing between already-correct formal contexts and those that need to be normalized to separate un-related word-pairs.

6.1.4 Implications and Association Rule Mining

This experiment did not take into consideration any attribute exploration methodology such as logical implications or association rule mining. While these methods are generally used for larger formal contexts, applying this experiment to a different data set and including such methodology could prove to generate better results.

6.1.5 Formal Concept Analysis in a Distributed Environment

Finally, in researching the In-Close algorithm for use in computing formal concepts, it seems that the algorithm is particularly suited for application in a distributed environment. During execution of the In-Close algorithm, each time a recursive call is made, that can be viewed as a separate process that can be executed on a separate machine. There has been some research in Formal Concept Analysis using the MapReduce model, but adapting this algorithm specifically remains a future consideration.

Bibliography

[1] Alberto Acerbi, Vasileios Lampos, Philip Garnett, and R. Alexander Bentley. The expression of emotions in 20th century books. *PLoS ONE*, 8(3):e59030, 03 2013.

[2] Simon Andrews. In-close, a fast algorithm for computing formal concepts. In *International Conference on Conceptual Structures (ICCS)*, Moscow, 2009.

[3] Gonzalo A. Aranda-Corral, Joaquín Borrego-Díaz, and Juan Galán Páez. Selecting attributes for sport forecasting using formal concept analysis. *CoRR*, abs/1107.5474, 2011.

[4] Cyril W. Cleverdon. The effect of variations in relevance assessments in comparative experimental tests of index languages. *Cranfield Library Report No. 3*, 1970.

[5] Stefan Evert. Corpora and collocations. In Anke Lüdeling and Merja Kytö, editors, *Corpus Linguistics: An International Handbook*, volume 2, pages 1212–1248. Walter de Gruyter, 2008.

[6] William B. Frakes and Christopher J. Fox. Strength and similarity of affix removal stemming algorithms. *Special Interest Group on Information Retrieval*, 2003.

[7] Bernhard Ganter. Two basic algorithms in concept analysis. In Lonard Kwuida and Bar Sertkaya, editors, *Formal Concept Analysis*, volume 5986 of *Lecture Notes in Computer Science*, pages 312–340. Springer Berlin Heidelberg, 2010.

[8] Charles R. Hildreth. Accounting for users' inflated assessments of on-line catalogue search performance and usefulness: an experimental study. *Information Research*, 2001.

[9] David A. Hull. Stemming algorithms - a case study for detailed evaluation. *Journal of the American Society for Information Science*, 1996.

[10] Julie Beth Lovins. Development of a stemming algorithm. *Mechanical Translation and Computation Linguistics*, 1968.

[11] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England, 2009.

[12] Martin F. Porter. An algorithm for suffix stripping. *Program*, 1980.

[13] Uta Priss. Formal concept analysis in information science. *Annual Review of Information Science and Technology (ASIST)*, 2006.

[14] Martin Reynaert. Parallel identification of the spelling variants in corpora. In *Proceedings of The Third Workshop on Analytics for Noisy Unstructured Text Data*, AND '09, pages 77–84, New York, NY, USA, 2009. ACM.

[15] Benno Stein and Martin Potthast. Putting successor variety stemming to work. In Reinhold Decker and Hans-J. Lenz, editors, *Advances in Data Analysis*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 367–374. Springer Berlin Heidelberg, 2007.

[16] Gerd Stumme. Concept exploration – a tool for creating and exploring conceptual hierarchies. In *In Proceedings Of The 5th International Conference on Conceptual Structures*, pages 318–331. Springer, 1997.

[17] Rudolf Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. Reidel, Dordrecht-Boston, 1982.

Vita

Graduate College
University of Nevada, Las Vegas

Guymon R. Hall

Degrees:

Bachelor of Science in Computer Systems Engineering 2001
University of Arkansas Fayetteville

Thesis Title: Using the Web 1T 5-gram Database for Attribute Selection in Formal Concept Analysis to Correct Overstemmed Clusters

Thesis Examination Committee:

Chairperson, Dr. Kazem Taghva, Ph.D.
Committee Member, Dr. Ajoy Datta, Ph.D.
Committee Member, Dr. Matt Pedersen, Ph.D.
Graduate Faculty Representative, Dr. Emma Regentova, Ph.D.