University of Nevada, Reno

Solving the Segmented, Static Database Paradigm By Means of Prismal Self-Organizing Maps

A thesis submitted in partial fulfillment of the
Requirements for the degree of Master of Science in
Computer Science

by

Salyer Byberg Reed

Dr. Gregory L. Vert/Thesis Advisor

May, 2007

ACKNOWLEDGEMENTS

I would like to sincerely thank my family – including my wife, my parents, my brother, and my grandparents – and my friends. Without their continued support and dedication, this monumental project would not have been possible. It is with great pleasure and honor that I acknowledge their nurturing and kind support in my endeavors.

I would also like to thank my committee members. My committee members have graciously bestowed their knowledge and time, furthering my education. It, too, is right that I should acknowledge their joint, vital contributions.

SOLI DEO GLORIA

ABSTRACT

Solving the Segmented, Static Database Paradigm By Means of Prismal Self-Organizing Maps

by

Salyer Byberg Reed

High dimensional datasets, such as text, are very often hard to cluster, and autonomous clustering of the dataset is even more taxing. Many algorithms have been developed and utilized, such as neural networks and classifier systems, endeavoring to cluster these datasets; however, most implementations require domain knowledge of the collection. A viable, more complete alternative to these systems is the self-organizing map.

The self-organizing map (SOM) intrinsically identifies structure and patterns in a high dimensional dataset such as a text corpus, or collection. Algorithmically, the SOM reduces the dimensional space of the collection, which is called quantization, and clusters it accordingly, creating observable relationships. This process is achieved through training the map over many iterations, and while the SOM is algorithmically robust, it lacks the capability to expand, or further cluster, once a map has been generated.

This novel idea presented in this thesis addresses the issue of the SOM's inability to expand; it is called the prismal self-organizing map (PSOM). In reality, it is inefficient – and sometimes impossible – to retrain a SOM if the dimensionality of the dataset is to be increased. As it is perceived, a more practical approach would be to create a new SOM. As such, the PSOM draws a parallel from one SOM to another by discovering strong correlations between the two maps. Once found, a connection is made, and this connection may be traversed, expanding the exploratory space.

For confirmation and analysis, the ideas fashioned in this thesis are applied to a segmented, static database. It is determined the PSOM provides adequate and fast query results for information retrieval.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1 – INTRODUCTION

The need for periodic backups of computer data is truly a necessity. Media types are not infallible, for they are known to fail, losing some or all data on the media, and computers are prone to break; thus, duplication of data is commonly practiced. From simple home computers to complex business networks, data is regularly dubbed onto various media types as a means of retrieval. However, over time, data can be diffused over many forms of media, making it almost impossible to retrieve, for – as mentioned – the data is dispersed, and iterating through the media forms is inefficient as media types have to be continually mounted and unmounted, consuming utility in the form of manpower.

Information Retrieval (IR) is a branch of Computer Science concerned with the storage and retrieval of data. IR receives an input, or query, from the user and proceeds to retrieve all documents that are relevant to the query, discarding documents that are not fitting to the query. The results, or documents, may be ranked according to the fitness of the summation of their respective weights. This process can be slow if many documents are to be scrutinized, which are located on different media.

Although storage of data on physical media is a minor task, information retrieval is nontrivial. Deeming a document's information as relevant is the objective of text mining. Once a document and its contents are classified, IR warehouses the information in a data structure used for data retrieval.

Data mining is the science of finding patterns in a set of data. Text mining is a derivative, or subset, of data mining; it finds patterns in text [23]. As opposed to the structured information used in IR, text mining sifts through unstructured data. When creating textual documents, grammar provides some uniformity among composition;

however, there are no definitive rules mandating text be structured to an exact fashion. It is the endeavor of text mining to transform this unstructured dataset into a structured dataset used for IR. As text mining is a subset of data mining, textual data, too, is predisposed to becoming dispersed over media and suffers from the same fallacies as its parent, data mining.

It is also noted that IR is most costly during runtime as it requires intervention by the user, consuming utility in the form of manpower. A selection, or query, is presented to the system, and a search is performed on the database. If a specific query does not match the assortment of documents and data, another query is performed on the remaining media until a match is found; this iteration can be monotonous, inefficient, and – as stated – expensive.

Other information retrieval systems are closed-loop systems. In addition to the traditional method of IR, including the input – the query – and the output – the results – the system introduces feedback [8]. Feedback converts the system into a closed-loop system; it captures output and processes it as input, refining the query and further output. Neural network [8, 9, 21, 29], fuzzy [X], as well as statistical systems[24, 25, 26] have been developed that successfully retrieve documents relevant to the user's query.

All of these methods described assume that the data is singular; that is, it can be obtained from one console without multimounting[1] media. However, data can be stored on multiple forms of media, and this media may not be singular. It is here that recalling an information's location and data is a problem.

When data is written to the media, it, perhaps, is physically stowed. This data is now *static* as it cannot change. There can be few if not thousands of backup media,

[1] Multimounting – The process of mounting, unmounting, and remounting the device.

depending on the application, regulations, or environment. Together, when all of these media, which are *segmented*, are combined, they form a segmented, static database, or an SSD [Figure 1.1].

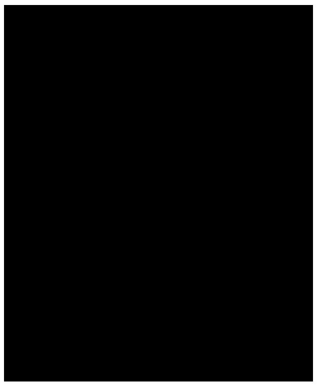


FIGURE 1.1: Representation of a SSD

This thesis proposes a technique of information retrieval and recollection for a segmented, static database by constructing a closed-loop system, a system with feedback, while maintaining a dynamic database of relevant information. Already widely known in computer science, the self-organizing map (SOM) provides an ample algorithm to cluster and quantize various spaces of high dimensionality, including a text corpus. SOMs have practical applications in text mining [15, 16, 19, 20]; nevertheless, once a SOM properly clusters a space, adding additional dimensions, or texts, becomes problematic; the traditional SOM lacks the ability to expand. The novel method developed in this thesis, deemed the Prismal Self-Organizing Map (PSOM), identifies correlations between two or more SOMs. Once identified, the PSOM creates connections between these paired SOMs; these connections may then be traversed to a region in the adjoining SOM, expanding the search space. Not only are the connections pointers to other SOMs, but they are also dynamic entities in that the region they point may be expanded to encompass more area.

In practical applications, the PSOM has proved to be an instrumental tool in expediting information retrieval in a SSD by lowering consumption of manpower through continual mounting of media. Utilizing statistical methods and text mining algorithms, documents in an SSD are perused for pertinent information and are classified and clustered in a PSOM. It is shown the PSOM, when used in conjunction with text mining, provides an adequate and accurate information retrieval process. More importantly, the PSOM provides dynamic, relational connections between SOMs, broadening a dataset space.

Chapter two is an introductory review of work related to information retrieval. It is an exploratory chapter, where one may become familiarized with the basic concepts and ideas applied in information retrieval.

Prior to clustering a dataset, in this case a text, a preprocessing phase must be applied to the dataset in interest; chapter three outlines the preprocessing steps performed before information is encoded into a map.

Chapter four introduces the very powerful concept – the self-organizing map – and its functionality, and chapter five is an extension of this subject with an emphasis of SOMs on a textual dataset. Chapter six identifies and discusses the various troubles associated with SOMs.

Chapter seven presents the idea of the prismal self-organizing map and its advantages. It investigates the creation and expansion of the PSOM. It also describes how various connections may be made between two or more SOMs.

Chapters eight and nine discuss the results and the conclusion, respectively. Finally, chapter ten is dedicated to future work in the realm of PSOMs and their applications.

CHAPTER 2 – RELATED WORK

Although not specifically designed to address the problems associated with SSDs, including dispersion of data over many media, many methods exist that assist categorizing and clustering a document's data. It is the endeavor of each method to discover the content, or composition, of each document; all methods are heuristic methods.

When parsing a document for relevance, it is hard to distinguish words, or terms, as relevant. To many, it is perceived that a document that contains like terms would be a good quantifier; for example, a document that contains many instances of the word "turtle," one could inference the document pertains to turtles.

By parsing the text a frequency of each term is tallied. Frequency is the number of occurrences, or instances, per document. When finished parsing the document, a distribution, either Gaussian or Poisson, is generated [Figure 2.1]. Because the principal concern is to ascertain the document's primary contents, the first n-terms within a minimum standard deviation are kept; each term describes the document.

FIGURE 2.1: A Gaussian distribution showing the mean and n-standard deviations from the mean

where: μ = mean, and

$n\sigma$ = n-standard deviations.

However, it is not plausible that all words in the distribution accurately describe the document. It is not the case with common vernacular; many words such as “the,” “a,” “and,” et cetera provide little disparity, or difference, from one document to the next. These words comprise the building blocks, or structure, of sentences; therefore, it is logical that one would have a large instance of these terms in a document. As such, deriving terms solely on the frequency of the term is an inefficient heuristic. A viable solution to this problem is to further limit the distribution, discarding frequencies greater than a specified distribution [Figure 2.2].

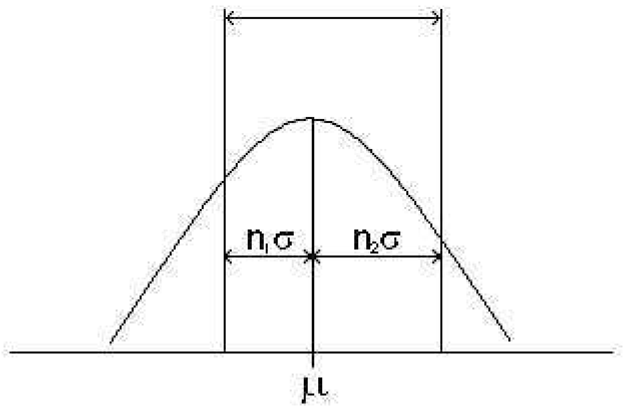


FIGURE 2.2: A Ranged Gaussian Distribution

where: μ = the mean,
$n_1\sigma$ = n_1-standard deviations, and
$n_2\sigma$ = n_2-standard deviations.

Nonetheless, this too presents problems. It is inadequate to discard high-frequency terms. Yes, it removes high-frequent terms, which most are common vernacular, but it possibly would remove high-frequent terms that accurately describe the document. With this current schema, the algorithm is unable to differentiate between terms that accurately classify a document and words that do not. Overall, this classification method is inadequate; however, there exists an efficient derivative of this heuristic in the classification of documents; it is the inverse document frequency, or IDF; another probabilistic model used in information retrieval.

An IDF system is fed a collection of N documents. Each document consists of many terms. When dissecting a document for contents, a frequency map is prepared. The frequency map is a collection of terms as well as the amount of occurrences in the document.

Using the frequency map, one may determine the significance a specified term contributes as it relates to the document, very similar to the previous method. This apportioned weight is deemed the term frequency.

$$tf = \frac{n_i}{\sum_d n_d} ,$$

where: n_i = the number of occurrences of term-i,
n_d = the total number of terms in the specified document, and

tf = the term frequency.

The term frequency – by itself – is similar in nature to the previous method. By itself, the term frequency provides no additional information concerning the document's contents. However, the inverse document frequency provides considerable information about a document's contents in relation to other documents in the set.

The inverse document frequency, or IDF, deems how relevant a term is in a specific set of documents. When calculating the IDF, once again a frequency map is created, yet this time, the frequency generated is the number of documents that contain the specific term [24, 25, 26]. Once the map is formulated, the IDF is calculated.

$$IDF(t_i) = \log \frac{N}{n_i} ,$$

where: N = number of documents in the set,
n_i = number of documents that contain the term t_i, and
IDF = inverse document frequency.

IDF is a probabilistic measure of a term's commonality; a term that is met continuously in various documents should be weighted less than a term that is found in few documents of the set. The denominator, or number of documents that contain the term, is the discriminator; large values in the denominator hinder large IDF values as can be seen in Figure 2.3.

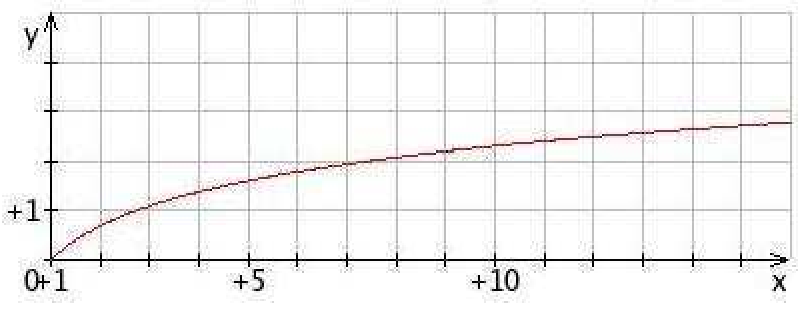


FIGURE 2.3: Log Graph Illustrating IDF

Combining IDF and TF – the synonym IDF-TF – one is able to easily ascertain valuable terms in a set of documents.

$$IDF - TF = \frac{TF}{IDF}$$

It is apparent that this definition will favor terms with high frequency in a document but low frequency in the set. The converse is also true. Therefore, IDF-TF provides invaluable information, regarding the contents of one document in relation to the set of all documents.

IDF-TF is not the only heuristic used in determining a term's relevance in a set of documents. In fact, Shannon entropy [15, 24] and PCA – principle component analysis – are other popular methods. In Shannon entropy, one determines and dictates information as relevant given observable trends in the data; it is a probabilistic measure of information [27]. Finally, PCA finds order, or patterns, in a high dimensional dataset – such as a document – and compresses it by reducing the dimensionality of the set, keeping the most relevant terms in the set [5].

Another popular method for text mining includes the use of genetic algorithms. Genetic algorithms are search algorithms derived from evolutionary principles. Used in searching large spaces, genetic algorithms perform recombination methods on individuals of a population, favoring more fit individuals [12, 28]. This bias provides an assimilation of desirable attributes, utilizing past information and conceiving new search domains. Also, for exploratory purposes of the space, randomness is habitually introduced into individuals during the next generation.

In genetic algorithms, individuals, or chromosomes, constitute the population of the system. Each chromosome represents a single entity of the communal population; it is an encoded representation of the entity. The chromosome, too, may be further divided. Each element that comprises the chromosome is called a gene. These are the basic exploratory elements of the genetic algorithm.

During each epoch, the population undergoes reproduction – usually in the form of crossover and mutation – where the population explores a designated search space. In basic crossover, two chromosomes are selected at random from the population set, which the selection is fitness proportioned. A fulcrum, or divisor point, is defined and the chromosomes swap elements along this fulcrum, creating new chromosomes, endeavoring to strengthen the fitness of the total population.

To further explore the defined space and regain fitness due to crossover, mutation is introduced. Mutation of a gene is a rather straightforward concept; given a defined probability, which should be rather low as to not introduce noise, randomly choose a number within a normalized range. If the number chosen is within the threshold of the probability, mutate, or manipulate, the gene of the chromosome.

Classifier systems incorporate genetic algorithms. During evolution of the population, the system learns syntactical rules, which are called classifiers [6, 10, 12, 28]. These classifiers mold the basis of exploration. The learning classifier system, or LCS, receives information from the environment and posts a message to the system. In reaction to posting a message, the classifiers proceed to predict the correct response through a competition, which is defined by their domain knowledge. In competing, if stimulated, a classifier will post its message to the message list, perhaps activating other classifiers in the process. Finally, after a winner is acknowledged, the system performs the appropriate action as deemed by the classifier, and the system is then rewarded or punished accordingly. To improve the response of the system, genetic algorithms are incorporated, introducing new syntactic rules through crossover and mutation.

Text mining may be achieved through classifier systems. In one instance, a classifier system peruses abstracts of papers for rhetorical structure [2]. As abstracts traditionally follow syntactic rules such as background, analysis, conclusions, et cetera, the abstract is fed to the system, a preprocessor identifies the subjects, which are fed in as input to the classifier system. The classifiers then proceed to create their own syntactic and rhetorical rules, allowing for classification and extraction of the document contents.

Neural networks provide ample solutions for information retrieval in terms of clustering. Specifically, the neural network is a classification method, much like the genetic algorithm, in that the system learns the algorithm. The neural network is composed of "neurons," or processing elements. Each neuron has a designated amount of inputs and a solitary output. Similar to biological neurons, the response of the neuron is proportional to the inputs. Also, associated with each input is a designated weight, either from the environment or the output of another neuron. Upon firing, the output is – traditionally – the summation of the inputs multiplied by the various weights of the inputs.

$$Output = \sum_{i=0}^{n} Input_i * Weight_i \ ,$$

where: n = number of inputs,
$Input_i$ = input value i, and
$Weight_i$ = weight associated with input i.

The structure of a neural network is a clustered set of neurons with interconnecting inputs and outputs. In the more traditional neural network, the network is comprised of three or more layers. While the first layer, or lattice, is deemed the input layer, the final interconnected layer in the network is the output layer and interlayers are deemed hidden layers. It is in the output layer that a decision is made, and, depending on the mode of the network, training occurs.

A popular training method amongst neural networks is the backpropagation algorithm. In this training algorithm, a desired output is known for a specific input. After introducing the input to the network, a conclusion is surmised at the output layer. The output layer is a compilation of classes, which one output node may accurately describe the document. As the output is already known, an error between the actual result and the result of the network is calculated. The network "backward propagates" the error through the nodes – adjusting each weight – and training commences again until the network is in concurrence with the actual output [29].

When backpropagating the error, the weights are first adjusted at the output layer, then the hidden layer, and finally the input layer; this explains the name of the algorithm. When adjusting the interconnected weights, a parameter called the learning rate dictates the rate of convergence for the system.

$$W' = W + \quad I_j ,$$

where: η = the learning rate,
δ = the error, and
I_j = the input.

If the learning rate is set too high, the system becomes unstable and never settles, or converges. If it is set too low, slow convergence, or no convergence, is observed in a specified amount of iterations. Once learned, the algorithm can accurately predict, or classify, similar, but unfamiliar, scenarios of its environment. Overall, this algorithm is considered extremely fast and an excellent predictor.

Many authors create a text retrieval system using a neural network. As with many textual neural networks [8, 9, 21], the input layer of the network is comprised solely of quantifiable terms; in some instances these terms are encoded in a binary representation.

In one instance, a system is created that incorporates two linkages in the neural network [22]. In the first linkage system, links are weighted according to the frequency of the term in the network and is adjusted accordingly during training. However, when new documents arrive, thus, new terms arrive, the system begins to forget previous learned rules; as such, the authors incorporate a second set of weights that are trained during the query, narrowing the search and increasing accuracy.

In another application, Crestani – much like his predecessors – consider the input layer of nodes the query and the output nodes the relevance of terms in the documents [8]. However, using probabilistic relevance feedback, the user is able to redefine the query, making the results more refined. In this experiment, each document is presented to the neural network and the error is then backpropagated through the network, training the neural network. When in runtime, the most significant document terms, which are the resultants of the output layer, are used to modify the query by appending the terms to the original query.

Finally, a group of researchers engineer their neural network a creative way by assigning each of the input nodes to the document terms and the query terms to the hidden nodes [31]. This perception of the network expedites training of the neural network, for the weights of the hidden layer are the only weights that are adjusted.

Text mining and classification may also be accomplished by implementing fuzzy set theory. In the realm of classical logic, an entity either resides in or does not reside in a particular group, or set; this is defined as the entity's membership. The domain of text mining encompasses this concept of classical logic; either a term accurately describes a document or it does not. For example, the term "salmon" strongly describes a document on "The Lifecycle of Pacific Salmon." However, the term "ocean" or "redd" may partially describe the same document. Fuzzy set theory accepts – and classifies – this partial membership [7, 32].

When classifying documents, the universal set, S, is the set of *all* terms in *all* documents, for each term, t, can be met from this set.

$$t \in S \ ,$$

where: t = a term of the universal set, and
S = the universal set; all terms in all documents.

Again, fuzzy set theory quantifies fuzzy descriptions; that is, it assigns numerical, or quantitative, values to qualitative descriptions such as “old,” “young,” “tall,” and “short.” In classical logic, again, one classifies elements as either belonging to or not belonging to a set; it is a square wave [Figure 2.4].

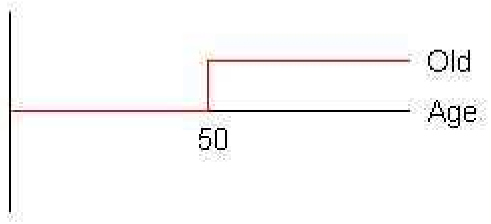


FIGURE 2.4: Classical Representation of Set Theory – A Square Wave

From this figure, there is a definitive, discontinuous line distinguishing individuals belonging to the set “Old.” An individual with an age of forty-nine years and 364 days is not in the set; however, an individual with an age of 50 years exactly is, in fact, in the “Old” set.

Classical logic can classify elements; however, it is straight-cut and can be inaccurate. Fuzzy logic, too, can classify elements. Each fuzzy set contains a regular set, S_f, as well as a membership function, μ_{sf}. On most accounts, the membership function is continuous on a specified interval. This interval is deemed the interval of confidence. It is noted that the interval of confidence is closed with the exception of infinity.

$$\underline{S} \leq S \leq \overline{S} \ ,$$

where: $\underline{S}$ = the lower bound, and
$\overline{S}$ = the upper bound.

In one fuzzy system, documents are clustered with respect to their index terms, which are the terms deemed relevant in a specific document using various statistics. These documents are then clustered, or categorized, accordingly using the fuzzy c-means algorithm. Once clustered, the system constructs a set of fuzzy rules: normalizing the centroids of each cluster, terms are sorted according to their weights. Later, term pairs are created from the cluster centroids, defining each classification rule. These rules bridge semantic connections between various index terms. Finally, these rules modify the query posed for retrieval by using fuzzy inferencing, which maps the query to a list of viable indexed terms.

When a query is presented to the system, the system calculates the similarity, using the cosine metric, between the query vector and the indexed weights. The index most similar to the query is the highest matched index; however, due to fuzzy similarity, a ranked order of other similar documents is presented as possible matches.

CHAPTER 3: PREPROCESSING OF A TEXT DOCUMENT

Before a text document is input to a system for processing, it must undergo a preprocessing stage. The preprocessing stage involves tokenizing, removing whitespaces, removing stopwords, and stemming words. Preprocessing segments the data of a document and helps reduce the dimensionality of a document, ensuring improved classification and clustering of data.

3.1 Tokenizing

A document is a corpus – a collection – of words, or terms. It is the purpose of the system to accurately decompose the unstructured words into identifiable features later used in classification and clustering. The foremost – and perhaps most trivial – phase is to tokenize the document. Tokenizing involves breaking the text of the document into tokens, where further processing may commence. Representation of each token is analogous to a dimension in the input vector; therefore, the dimensionality of each document will be…

$$\dim(doc) \leq n \ ,$$

where: n = the number of tokens in the document.

As tokens may be recurring in each document, the dimensionality of the document does not increase with each duplicate instance of an identifiable token. For example, a document consisting of nothing more than the word "circle" would have only one dimension.

One should note that computers are, ultimately, linguistically unintelligent. Although relatively simple for an individual, who is accustomed to the rules of the language, to easily decipher and tokenize a document, computers are not so fortunate;

they are prone to errors. It is accurate for an entity to easily tokenize words according to whitespaces. Whitespaces are considered to be spaces, newlines, tabs, periods, quotes, dashes, apostrophes, et cetera. The whitespace is considered a delimiter, for it concisely defines the boundaries of the token, separating and distinguishing one token from another.

Again, the subject of tokenizing is perceived to be simplistic, but a computer is not so providential. In some instances, a delimiter may be considered part of the token. Computers, unless specifically instructed, cannot ascertain if a symbol in the language should be considered part of the token or is a delimiter. Take, for example, the word "O'Reilly." In this example, the computer must not delimitate the word into two separate tokens, for "O'Reilly" is a proper noun and must be treated as such. Another paradigm to the computer is a possessive form of a word. Again, for example, take the word, "Benjamin's." Is the computer to tokenize this word into two tokens – Benjamin and the letter S, discarding the token S – or is it to keep it in its form and create a solitary token: Benjamin's? Unknown to the computer is the semantic of this word. If the subsequent word of "Benjamin's" is "book." One would assume the word should be divided into "Benjamin" and "S," disposing of the token "S." Conversely, suppose one were talking about going to the restaurant named "Benjamin's" or going to "Benjamin's" place of residence, neglecting to add "house," "residence," "place," et cetera. These would still create a valid sentence, or structure, in the English language; as such, one would consider the apostrophe as a piece of the token.

Other paradigms arise with other instances of whitespaces. Another example includes the use of the dash. In the instance of a phone number, the dash should be considered part of the token. On the other hand, a dash in the independent clause, "the wolf - an ancestor of the dog - is wild," should be considered a delimiter and handled as such.

To combat the problem of tokenizing, the algorithm displayed in Figure 3.1 actively peruses each document. In the algorithm, an alphabet of whitespaces, excluding the apostrophe, is used in tokenizing.

```
DEFINE WHITESPACE : {" ", "\n", "\t", ".", ";", ...}

$pos = 0$

WHILE $document[pos]$ NOT EQUAL EOF
{
    $begin = pos$

    WHILE $document[pos]$ NOT IN WHITESPACE
    {
        $pos = pos + 1$
    }

    WHILE $document[pos]$ IS IN WHITESPACE
    {
        $pos = pos + 1$
    }

    $token = document[begin..pos]$
    $i = StringLength(token)$

    WHILE $token[i]$ IS IN WHITESPACE
    {
        $Truncate(token, i)$
        $i = StringLength(token)$
    }

    $aposLoc = ApostropheInToken(token)$
    IF $aposLoc$ DOES NOT EQUAL -1 THEN
    {
        IF $token[aposLoc + 1]$ DOES NOT EQUAL NULL AND
        $token[aposLoc + 1]$ EQUALS 's' AND
        $token[aposLoc + 2]$ EQUALS NULL THEN
        {
            $i = aposLoc;$
        }
    }

    $Truncate(token, i)$
}
```

FIGURE 3.1: The Algorithm Used in Tokenizing Documents

To summarize the algorithm, once a token is identified in a document, a post-process is performed on the token. This post-process involves removing other delimiters such as excessive spaces, commas, periods, quotes, and other symbols used in the predefined delimiter alphabet. This processing stage also includes removing the troublesome apostrophe. Once an apostrophe is encountered in the token, the program checks the following subsequent character. If the subsequent character is an “s,” *and* it is the last character in the token, the apostrophe it is removed along with the character “s.” Also, if the last character in the token is an apostrophe, the apostrophe is removed from the token. This simplistic algorithm removes the dilemma associated with apostrophes. In “O’Reilly,” the apostrophe is preserved while in “Benjamin’s,” the word is truncated to “Benjamin.”

It is apparent that this algorithm is not lossless, for the algorithm truncates information it deems useless. It is true that information will lost, but the amount of information lost is qualitative and discretionary. Nevertheless, during quantization of the documents, information, too, is truncated from the set. To ensure uniformity amongst the data and the query, the query, too, should adhere to the rules defined by this algorithm. Only when the query and data storage are constrained to the same rules will they produce similar results. Again, consider the example, “Benjamin’s.” If deemed to be a descriptive word of the document, the information storage algorithm will truncate “Benjamin’s,” resulting in “Benjamin.” Unbeknownst to the user – as it is done behind the scenes – the query, too, will truncate the word, resulting in a match, creating homogeny between the systems.

3.2 Stop and Stemming Words

Prior to processing and training the SOM, traditional methods of classification and clustering include removing unwarranted information from the set. In text classification and clustering, there are two forms of superfluous information contained in text: stop words and stemming words [3].

Stop words, in general, are words that provide no insight into the prediction or classification of text and its content. Stop words traditionally include words such as *a*, *the*, *it*, and other common words of a particular language. Removing these stop words, reduces the dimensionality and complexity of the language while preserving content of the textual corpus.

Many methods exist to remove stop words. The foremost and effortless method involves discarding the stop word tokens. Generating a dictionary of stop words prior to classification is seemingly simple; any tokens that match an element within the dictionary are, ultimately, discarded.

Another method includes morphization. One should note that morphization might only be applied to specific stop words as will be apparent below [30]. When various stop words are encountered, such as *it*, one knows the word – *it* – is to predicate an existing subject. Through syntactical rules of the language, one should easily substitute the subject in for the stop word. Although simplistic, problems exist in "loose" languages. For example, consider the following sentence: "The wolf ran through the forest; it was covered in snow." While it is syntactically correct, there is ambiguity in the sentence. Is the wolf covered in snow, or is the forest covered in snow[2]? There is no definitive algorithm to correctly identify the object in question, for the language is loose. If the object is incorrectly identified, then the results from classification and hence the mapping is skewed. Therefore, special consideration should be considered when morphing words as to not incorrectly identify the stop word.

[2] In the English language, it is grammatically correct to consider *it* to be of the last subject, but not everyone follows the rules of grammar; as such, the English language is "loose."

Morphing in a strict language is more permissible. In a strict language, ambiguity is removed from the document as the language must follow a predefined set of rules, or grammar rules. If the document does not adhere to these rules, then the document is invalid and should not be added to the set to be classified.

Once stop words are removed from the data set, the lemmatization or stemming process is performed on the set. Although partially addressed within the implementation of this thesis, lemmatization can drastically improve the accuracy of the map. Lemmatization involves the idea of reducing the size of the word set by increasing the frequency of each term in the set. Almost all verbal languages – especially romantic languages – words have a root word [4]. When changing the context in which a word is used, many languages need to append or remove a few letters of the alphabet to the root word. For a simplistic example, consider the root word "play." To change this word into past tense, one would append "-ing" to the root word, creating "playing." To change to present tense, one would append an "-s," producing "plays." All of these words have a common root – play. Therefore, if these words were ever encountered while parsing a corpus, one could reduce the size of the set by removing each individual word and increase the frequency of the root word, "play."

Another – more drastic – form of lemmatization exists in that not only are suffixes removed, but prefixes are removed as well. For example, the word "readjust" could be reduced to "adjust." A combination of removing suffixes and prefixes can be applied as well; "readjustable" would be reduced to "adjust" as well.

One issue must be addressed when lemmatizing, or, once again, information will be skewed. The issue is simple; some words cannot be easily stemmed to their root word. In these cases, these words will increase the size of the set, establishing false, or repetitive, groups. Consider the word “ate.” It is the past tense of the word “eat;” hence, the root word is “eat,” but stemming is impossible if not for a local dictionary that the algorithm may reference.

In contrast, lemmatizing is not as simplistic as perceived. In reality, words in many languages can have many definitions. It is a complicated task – if not impossible – for a computer to easily discern the difference between uses. Examine the following two independent clauses: “the boss fired the employee,” and “he fought a lot of fires.” After lemmatizing, it is apparent that both clauses contain the single root word, “fire.” Unbeknownst to the computer, the words used in each clause are used in different contexts; as such, when generating a frequency distribution, the term “fire” should occupy two separate areas in the map and, perhaps, distance themselves from one another, depending on their context, for a “wildfire” is traditionally not correlated to the act of “firing” an individual.

It is simple to see the advantages of morphing and lemmatizing; they reduce the size of the set by increasing the frequency of each element in the set. Therefore, when a map is assembled, repetition and false groups are minimized.

CHAPTER 4 – FOUNDATION OF THESIS: THE SELF-ORGANIZING MAP

It is the purpose of this thesis to address the issue of a segmented, static database, or SSD, which all the data is dispersed between media. Retrieving specific data from a SSD can be daunting task:

- It is time consuming,
- One may not know the specific location on the media where the data resides,
- Worse, one may not know the specific media that contains the data.

The entire scenario is synonymous to finding a needle in the haystack.

Information retrieval for databases is the process of recalling data; its concern is the systematic storing and extraction of data. Current information retrieval systems receive queries from a user and peruse the database for specific matches. Similarly, although the data is dispersed across many media, the collective, nonetheless, is a database, for it is a set of data.

Most databases consist of many documents, and a query is comprised of many terms. Referencing the queried terms against the contents in the documents is computationally expensive. If an index of terms is formulated, a parser can parse the document, searching for matches. For one query term compared to all elements in the document, the asymptotic runtime is O(n), for the search is linear. However, it is not the asymptotic runtime that is imperative to the system. SSDs posses the redundant task of mounting, searching, and unmounting media each time a query is initiated. One must note that many databases contain many documents and many documents can contain many elements, or terms; therefore, querying a database of great magnitude can be fruitless; there is the possibility there will be no successful retrieval. This is the crux of the SSD problem.

The problem stems from writing textual data to media. The information is retained; however, the user may forget file name, the file's contents, and file's location in an assortment of media. As a result, current information retrieval methods are faulty and time consuming, for – as stated before – they are costly during runtime; included with runtime is the repetitive task of mounting media.

A self-organizing map, or SOM, is an extreme derivative of the more traditional neural networks such as a backpropagating network. While traditional neural networks require training data to contain complete information about the inputs, its characteristics, and desired outputs, SOMs only necessitate input. Hence, a SOM – unlike its predecessor – is an algorithm for unsupervised learning; it is the ability of the SOM to learn based on the similarities, or dissimilarities, of the set data. This inherent learning feature of the SOM is called clustering [17, 18].

The foundation of clustering as well as SOMs finds itself in the realm of prediction. Prediction is the science of accurately forecasting the future based on similarities of past experiences and characteristics. For prediction to be accurate as well as consistent, patterns in the past must be identified that apply the future cases.

It is inaccurate or impossible to predict the future if the characteristics differ greatly from that of the past. In fact, machine learning does not have rational; that is, it cannot reason or reach a new conclusion from new information. Therefore, for a system to accurately predict – or in this case, classify – it must possess and experience outcomes from prior information. Only then can a system properly classify future instances of unknown results.

Clustering, in terms of SOMs, involves categorization of data based on the similarities, or likeness, of the input. One must remember that traditional neural networks have two characteristics: prediction or classification. For purpose of data retrieval, one is more interested in classification. Classification is the process of grouping data into various *predefined* groups based on experiences from the past. Again, SOMs are a derivative of traditional neural networks. As opposed to the more generalized category of classification, SOMs cluster data. Clustering is similar to classification; however, data is segmented into nonpredefined groups. These groups are organized according to similarity of the input data.

SOMs also differ from their counterparts – such as multilayered feed-forward neural networks – in terms of their architecture. One should recall that traditional neural networks contain many layers: an input layer, and output layer, and an arbitrary amount of hidden layers. Generalized SOMs, on the other hand, are comprised of two layers: an input layer of nodes and a classification layer of nodes, or lattice layer [Figure 4.1].

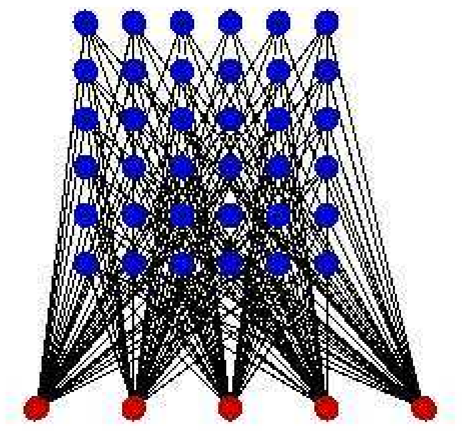


FIGURE 4.1: A Fully Connected SOM

As depicted by the figure, each input node in the input layer is connected to each node in the output lattice; for this reason, the node is deemed fully connected. Much like a traditional neural network, each connection, or edge, has a weight associated between the two entities. SOMs are feed forward neural networks in that information is “fed” into the input nodes as these nodes further cascade information to other layers. As stated, SOMs only have two layers, so the input nodes flow into the classification lattice. It is the purpose of the SOM to reduce the dimensions of the input vectors; this is called vector quantization [17].

Initially, each weight between the input nodes and nodes in the lattice are randomly assigned values. As data are presented to the system, the network is fed an input vector. Therefore, as each node in the input layer is stimulated so, too, are the nodes in the output lattice. SOMs are competitive in nature. Much like classifier systems, each node in the SOM competes to be the best-matched unit, or BMU. The BMU is a node that most closely resembles the input data. During stimulation, each node calculates its similarity, or distance, to the input vector. Traditionally this measure is the Euclidean distance…

$$d = \sqrt{\sum_{i=0}^{n} (V_i - W_i)} \ ,$$

where: V_i = the vector component, and
W_i = the associated weight.

The member that is most similar to the input – that is, the member that has the minimal distance – is deemed the BMU and is rewarded [17].

Reward is the base for learning. In all, learning, for a neural network, is the adjustment of weights in the system. After a BMU has been identified for a specific input, a reward is applied. However, the reward is two-fold; not only is the BMU rewarded, but the BMU’s neighbors are also rewarded. The structure of the lattice can be depicted as a two-dimensional array, and learning transpires in clusters. For that reason, each neuron in the lattice has a distinct neighborhood [Figure 4.2].

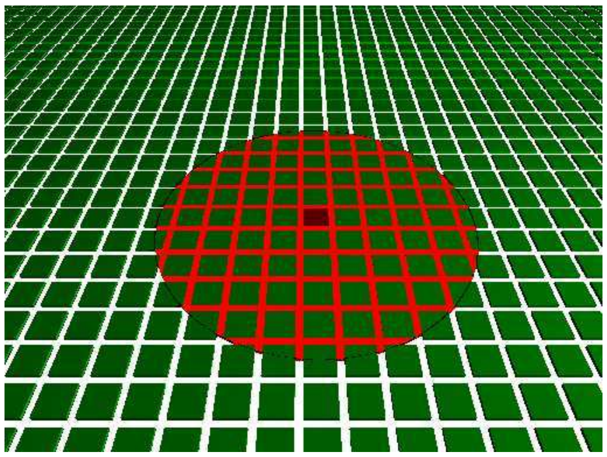


FIGURE 4.2: A Neighborhood of a BMU

When rewarding, it is surmised the BMU should receive the most reward, and neighbors should receive reward determined by their proximity to the BMU; neighbors closer to the BMU received far greater reward than those further from it. This is an example of diminishing propagation of rewards amongst neighbors and shall be known as the ripple effect [Figure 4.3].

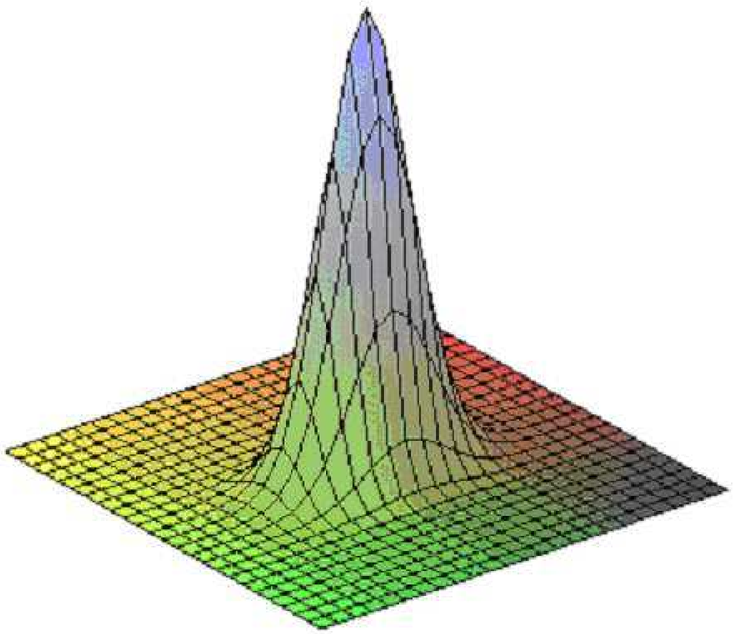


FIGURE 4.3: Ripple Effect

Like traditional neural networks, when determining the amount of reward, one traditionally uses a predefined equation for adjusting the weights. Also, embedded within each weight equation is a learning rate. The learning rate – in general – controls the rate of convergence of the system. High values tend to converge the system rapidly while with lower values the system slowly converges. Finally, it is advisable to consider the ripple effect when defining an equation. Again, one should reward individuals closer to the BMU than those that are further away; classical examples include exponential decay and inverse distances.

It is also permissible to have considerably large, dynamic neighborhoods in early epochs of the algorithm while slowly contracting the neighborhood during further iterations. In fact, Kohonen, the architect of SOMs, even proposes going as far as setting the neighborhood size to just the set of {BMU} upon the final epoch. This large neighborhood and contraction ensures initial epochs do not inundate the global populace with false optima. Instead, the sizeable neighborhoods – at first – encourage global unity among nodes. Later iterations include a thinning neighborhood; this contraction ensures a focused resolution of each group.

Before training commences, the weights amid the neurons are assigned random values. Once the system is fully connected and the input and output nodes are defined, training commences. An input vector is fed into the input layer and the BMU is identified. During early epochs, or iterations, the learning rate as well as the neighborhood size should be larger than that of later epochs. As further iterations are completed, adjusting these parameters ensures localized and accurate clustering of the data set.

Unless specifically built into the evaluation, or weighting, algorithm, it is impossible to define the amount of clusters the SOM will produce. As the SOM clusters data from the set, clusters begin to form, expanding and contracting, until map optima are obtained [Figure 4.4].

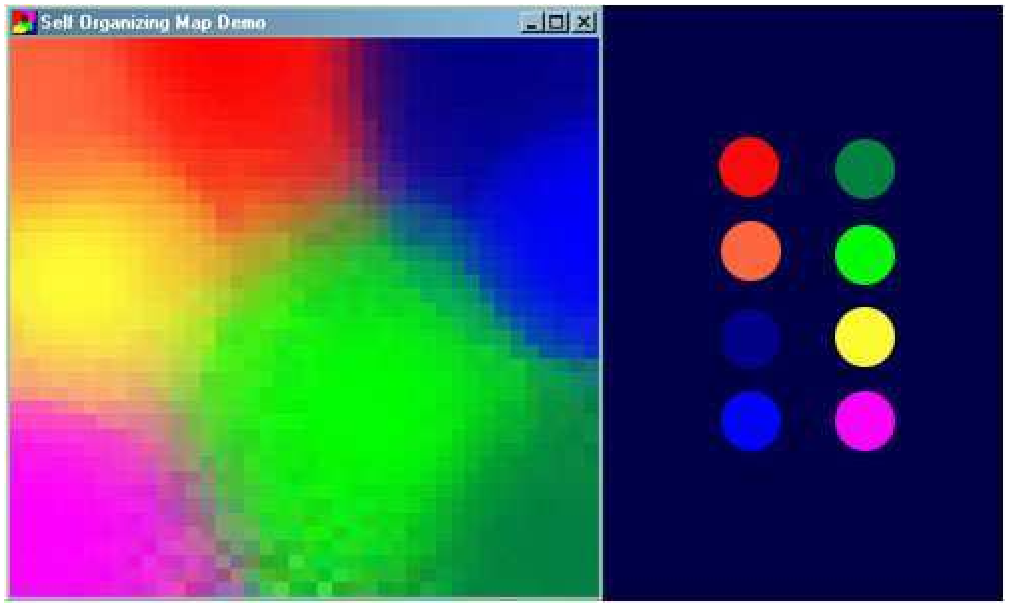


FIGURE 4.4: Map Optima [1]

These map optima contains all the clusters obtained by the SOM algorithm; quantization is complete. Especially important to clustering is the result of map optima. If the map contains a large amount of relatively small clusters, the map guarantees uniformity among clusters, for the vector inputs differ significantly from the other input vectors that are fed into the system; on the other hand, a small number of large clusters proposes diversity amongst the clusters.

In regards to predetermining a map's size – as is noted by Kohonen through observation – a respectable size for the map shall be approximately ten times the size of the input vector; however, this number is variable and open to the discretion of the creator [17]. Also, the amount of iterations is open to interpretation and the judgment of the creator, but traditionally training ceases when saturation of the map occurs; that is, no relevant change in the map nodes can be observed.

CHAPTER 5 – CREATING A SELF-ORGANIZING MAP FROM TEXT DOCUMENTS

As mentioned in chapter three, preprocessing includes tokenizing the document and removing all stop words, stemming words, and whitespaces from the document utilizing the methods of morphing and lemmatizing. Afterward, for each remaining term, the IDF-TF is calculated [16, 19, 20]. Once calculated for all documents in the set, thinning was performed.

Thinning utilizes the IDF-TF. Normalizing the IDF-TF for each term, a mean and a standard deviation for IDF-TF is calculated from the set. For all terms with an IDF-TF value below or above a specified standard deviation, they are omitted from participating in clustering and are discarded. A collection of relevant terms now remains, accurately detailing a document's contents. Following thinning, a SOM is created from the corpus of texts as previously described.

The nodes of the output lattice are randomly initialized, and a weight is only adjusted if rewarded; specifically, the node is the BMU or within close proximity of the BMU. The traditional equation of adjusting a node is as follows…

$$w_i(t+1)=w_i(t)+h_{ci}(t)\big(x(t)-w_i(t)\big) \ ,$$

where: w_i = a specific weight of the node,
h_{ci} = the neighborhood function of the node,
x = the vector input, and
t = a discrete-time integer [16, 17].

The neighborhood function, h_{ci}, is proportional to the learning rate of the system as well as the radius at a specific time. Traditionally this is a Gaussian function defined as…

$$h_{ci} = \quad (t) * \exp\left(- \frac{\|r_c - r_i\|^2}{2 \quad {}^2(t)} \right),$$

where: α = the learning rate,
||r$_c$ - r$_i$|| = the distance from the BMU and the node of interest, and
σ = a radial function.

In this equation, it is noted that the learning rate and radial function, decrease with time.

CHAPTER 6 – PROBLEMS WITH SELF-ORGANIZING MAPS

SOMs are very good at clustering data; however, they are severely impaired when the map needs to expand, or grow. When creating a SOM, the vector space of the document set is clearly identified, and each of the input nodes are fully coupled with each node in the output lattice. One must remember that the number of dimensions is equal to the amount of unique words identified by the preprocessor; the inputs of the system are static. Therefore, SOMs are static, singular entities; that is, when a SOM is created and finalized, it is exceedingly difficult to append to the SOM. Consider, for example, a finalized SOM representing the three basic additive colors: red, green, and blue [Figure 6.1].

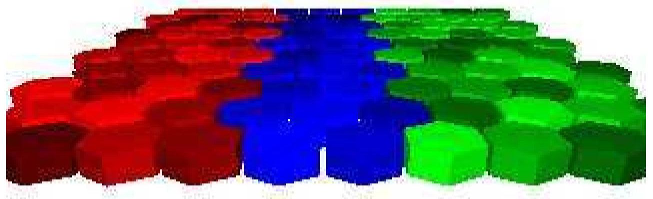


FIGURE 6.1: An RGB SOM

Suppose, after much deliberation, another element were to be added to the SOM; in this case the vibrant color yellow, which is defined – in the standard 8-bit coloring system – as RGB(255, 255, 0). From inspection, it is noted that this permutation of yellow contains all red as well as all green on the spectrum. A logical conclusion would be to associate the yellow near the red and the green yet away from blue; however, the red and the blue occupy different trisectors of the map. Inserting the yellow between the two – in close proximity of the blue elements – creates a false impression that yellow is related to blue.

To solve this paradigm, one has two solutions. First, a complex recombination algorithm would be needed to reorganize the map while preserving the relationships and minimizing errors among elements. This algorithm is not guaranteed to be lossless. The second possibility is to retrain the map. With the initial set of information, append, or unionize, the new element with the set, creating a new set. From the new set, retrain and create a new map.

In the SSD, one is not as lucky as to retrain the map, for information is diffused among various media and is cumbersome to retrieve. The redundant, constant remounting of media to retrieve sparse information is impractical and inefficient; as such, there is a perceived need for constructing an SOM that is dynamic in nature; it must be able to expand its dimensions and bounds, while creating and preserving relationships among the elements contained within the map.

It should be noted that it is possible to append to the map. When there is a desire or a need to append to the map, one should present the new element to the map, finding the BMU, or best-matched unit. Once the BMU is identified, a pointer may be inserted, or attached. Employing this method is acceptable only for high-dimensional, large maps. Low-dimensional and small maps are not as fortunate, for as the map is sparser, false relationships arise. This was illustrated in the aforementioned color SOM example.
If large amounts of data are required to be added to the SOM, this method of appending to the map is also discouraged, for the new data elements cannot accurately be described by the statistics of the old training data. Therefore, retraining would be required, or a new, accurate method of appending to the map would be needed.

CHAPTER 7 – INTRODUCTION TO PRISMAL SELF-ORGANIZING MAPS

7.1 Prisms

Prisms, as defined in optics, are geometric objects capable of refracting light in various directions. It must also be noted that the color white is a multitude of colors intermingling with each other at different frequencies. As a white light permeates the prismal object, the speed of the light is reduced and ultimately filtered according to its color, or frequency [Figure 7.1].

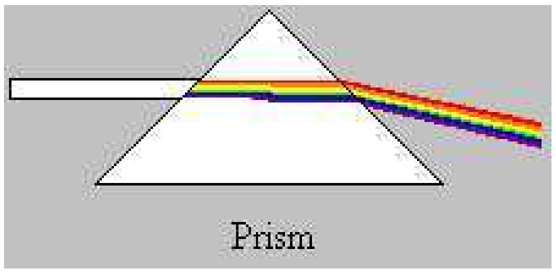


FIGURE 7.1: A Triangular Prism, Refracting Light

The same is true with textual information; expressions and variables are combined, creating a unique variation of logical and structured sentences. These sentences, too, are pooled, creating paragraphs. This process is iterative ultimately producing a structured and syntactical document.

One may consider a self-organizing map a prismal object; documents consisting of various topics and phrases are grouped into a set and presented to the SOM. This is analogous to the white light, for the documents are uncorrelated at the time of presentation. It is the objective of the SOM to filter and cluster these documents according to the structure of their contained data. This is similar to the prism's natural ability, which is separating the light according to color.

It was Sir Isaac Newton who first conceived the idea of placing prisms in series and observing their result. Through inspection, however, it was determined that the colors were, in fact, not further diffused. However, one may apply this simple principle to SOMs. Moreover, this is the basis for PSOMs. In brief, a PSOM is a collection of SOMs in series where each SOM maps itself to the next SOM in the series through associations. When there is a strong correlation between elements in different SOMs, a connection is made between the two maps, corresponding to these two elements in the maps and potentially their neighborhoods.

7.2 Prismal Self-Organizing Maps

Before the theory of the PSOM is introduced, one must note it is the endeavor of this thesis to cluster documents according to the framework of their enclosed data; as such, the description of the PSOM shall be aimed at document classification. It is the endeavor of the PSOM to assist in locating data in an SSD. Nevertheless, there are no boundaries to the utilization of PSOMs; the PSOM is just as diverse as its predecessor, the SOM, and shows promising ventures in many fields of science and clustering.

The Prismal Self-Organizing Map is a collection of SOMs in series. It can be perceived as a map that is “mapped,” much like a linear transformation, onto another map; this process is iterated throughout the chained SOMs [Figure 7.2].

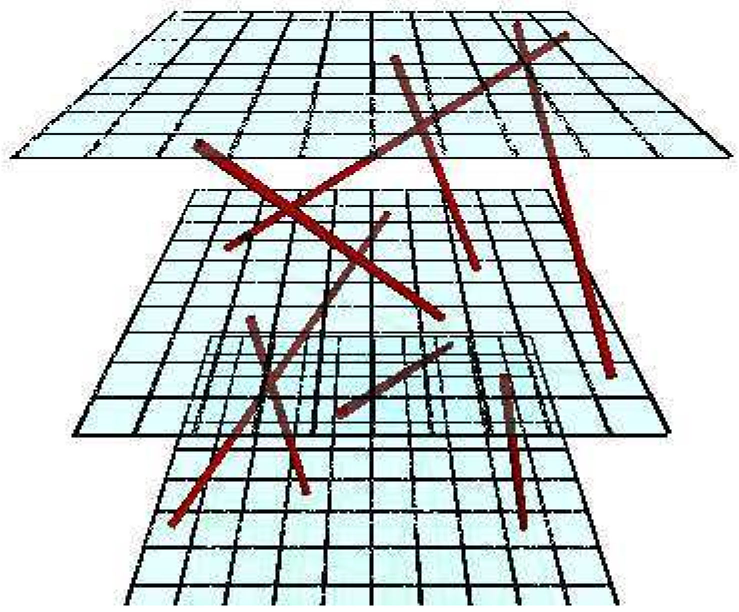


FIGURE 7.2: A PSOM with Various Connections

Conception of a PSOM is simplistic: create a SOM. This is the base of the PSOM. As it is the pedestal of the PSOM, each supplemental layer, or SOM, will be appended to this map. As depicted prior, a preprocessing phase is applied to the data set, including morphing and lemmatizing. Following the preprocessing phase, the TF-IDF phase is employed, carefully selecting and removing elements from the set. Finally, the remaining elements in the set are fed to the SOM algorithm for clustering and positioning within the map.

Intra, the documents that were prepared for backup as well as the SOM are written to the specified media. If the desire exists and the media permits it, the SOM, too, may be written to the media. Therefore, a local mapping of the contents can easily be accessed should the need arise.

The SOM that was recently created, describing the documents on the media, is then appended to the PSOM, which is a collection of SOMs each describing a separate media. When affixing the new SOM to the PSOM, correlations are uncovered relating maps to one another. Upon completion of correlating the elements, the PSOM provides accurate and detailed information about documents, including a document's location on a specified media as well as documents closely related it, making a query fast and retrieval easily accessible.

At the next instance of backing up documents to media, the same process is iterated and the resultant SOM is appended to the PSOM, again, expanding the PSOM and finding new correlations between documents and media. Ultimately, the PSOM is a dynamic, flexible structure, expanding and contracting as deemed by the system or the user.

Briefly, the benefits of PSOMs include relational mappings between SOMs. As already described, a PSOM finds relationships between different layers – or SOMs. Once the relationships are found, the PSOM connects interrelated elements with connections. The connections are the key to establishing parallels between seemingly noisy, or random, elements in the SOMs.

Another added benefit of the PSOM is its ability to dynamically reconstruct the statistics of a SOM layer. Suppose, for instance, that during the creation of a SOM two elements – as described by the input statistics – are perceived as independent. As they are perceived as independent, it is probable the SOM will place padding between these two elements [Figure 7.3].

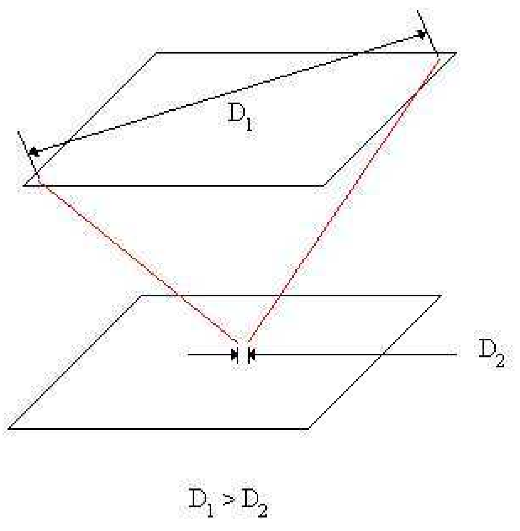


FIGURE 7.3: Seemingly independent quadrants are correlated with the aid of connections; D_i is the distances between the two quadrants

In reality, these two elements may, in fact, be closely correlated, but when presented to the SOM algorithm for classification, the statistics and inputs do not properly describe the relationship. In continuing the example, suppose another SOM is constructed. Upon running the PSOM algorithm, a relationship – or a bridging connection – is found between the two elements with the aid of another SOM in the PSOM. Because of this bridging connection, the two elements, which reside in the same SOM, can be correlated and provide accurate statistics for querying.

7.3 Growing the Prismal Self-Organizing Map

Once the base SOM is created, other SOMs are appended to the PSOM by ascertaining the similarities as well as the dissimilarities between the structures. If a similarity is well pronounced – above a specific threshold – a connection is constructed, connecting the two elements of different SOMs and their proximate neighborhoods. The contrary is also true; dissimilar elements should support no connections.

When the next SOM is created and is appended to the PSOM, the same process is iterated; connections are made between the elements. Each SOM is paired against that with each SOM already residing in the PSOM, and connections between the elements are created if they exist.

7.4 Neighborhood Weights

Creating connections between elements in different SOMs is not sufficient. If it were, the PSOM would contain very little additional information, paralleling different SOMs. Much like the neighborhoods created in the original SOM, connections must have neighborhoods. As the connection has an origin and a destination, each terminal, too, must support a neighborhood. Therefore, connecting neighborhoods are correlated and adequately describe the statistics of the PSOM. It is the purpose of clustering to adequately define these neighborhoods for association. The neighborhood of the connection may be defined by its simple neighborhood equation. Again, a multitude of metrics exists in determining the neighborhood.

Perhaps the simplest method would be to define a radius for the element. All neighbors within the radius are perceived to be closely related to the element; as such, an association is made between the two elements on the terminal ends of the connection. This neighborhood relationship is also a simple distribution, either Gaussian or Poisson; the further away from the mean, or terminal connection, the lesser the association between elements [Figure 7.4].

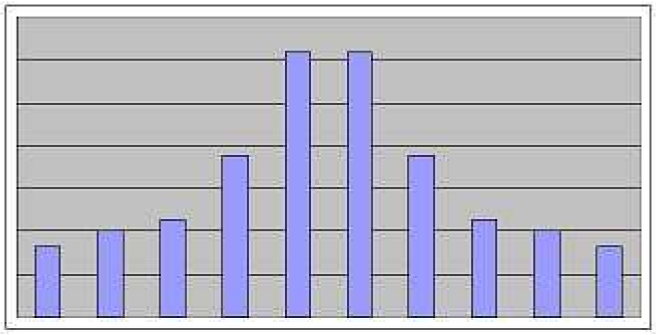


FIGURE 7.4: Poisson Distribution

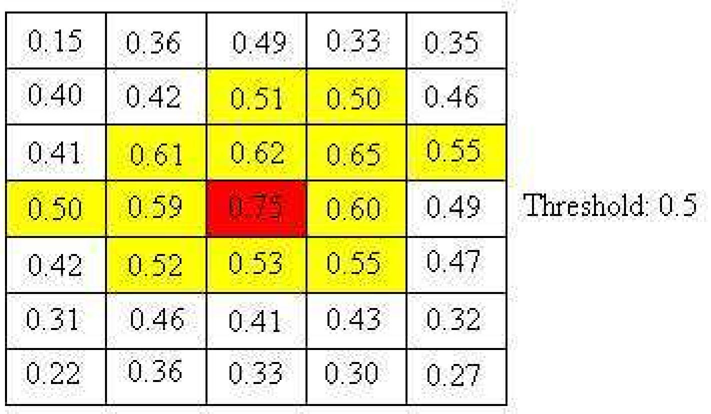
Another, perhaps more appealing, metric for determining the neighborhood is the Similarity Amongst Neighbors method. In this method, the weights, which are originally connected to the input lattice, are used in determining the similarity. Starting with the terminal element of the connection, all coupled nodes are searched for their “similarity.” Given a specified error, if the Euclidean distance measure between the terminal element’s input weights and the residual element’s input weights falls within the designated tolerance, the node is considered a neighbor. Consequently, the connecting nodes of the newly found neighbor, too, are searched for their similarity. This process is iterated until no more neighbors are found in the underlying SOM. It is interesting to note that this method may produce irregular neighborhoods; however, each neighbor within the neighborhood is similar to that of the terminal node, defined by the specified error.

7.5 Creating Connections

It is the purpose of PSOMs to find similarities, or differences, between SOMs and create connections between them. Creating connections, or associations, between one SOM to another is a rather mundane task. Very similar to its predecessor, the SOM, the PSOM creates connections by identifying the BMU in the destination SOM using varying heuristics.

As each node in the SOM is fully connected to the input lattice, each node may be described by the weights of these connections. Therefore, it is only obvious that these weights and input nodes be used to bridge two SOMs. The algorithm is simple; for a solitary node in the destination SOM, find the BMU in the source SOM.

For each connection made, the terminals, too, have neighborhoods. It is noted that the neighborhoods may be radial, like a SOM, which is easy to comprehend. However, the neighborhoods may also be indistinct, or nonsymmetrical. To create an irregular neighborhood, start with the BMU. For each adjacent node, compare the dimension to that of a set threshold. If the dimension value is within tolerance, the node is considered a neighbor and each of its adjacent nodes are recursively verified for membership. An example is illustrated in Figure 7.5, where the red square indicates the BMU, and the yellow squares indicate neighborhood membership. Nodes that are transparent lie outside the neighborhood.



0.15	0.36	0.49	0.33	0.35
0.40	0.42	0.51	0.50	0.46
0.41	0.61	0.62	0.65	0.55
0.50	0.59	0.75	0.60	0.49
0.42	0.52	0.53	0.55	0.47
0.31	0.46	0.41	0.43	0.32
0.22	0.36	0.33	0.30	0.27

Threshold: 0.5

FIGURE 7.5: Irregular Neighborhood

It must be noted that if neighborhoods are identified by one dimension, traditionally the neighborhood will be radial as the SOM algorithm dictates the neighborhood. On the other hand, two or more dimensions used as identifier begin to pull and stretch the neighborhood, for they act as poles, making the neighborhood more irregular.

7.5.1 Euclidean Method

One possible heuristic in identifying the BMU would be to calculate the Euclidean distance between the target node and each node in the receptive SOM. If possible, one should also normalize the information being employed. Once calculated, the node most similar is deemed the BMU, and a connection is made. One must note that the dimensionalities for each input lattice for both SOMs are near limitless; therefore, this heuristic only works well when the dimensional space, of the entire document set, is rather small and contained. On the other hand, if possible, it will work if the Euclidean distance is normalized and compared to a threshold; for all values exceeding the threshold a connection is made and vice versa. Once a connection is made, a neighborhood may be constructed and perused for useful information.

7.5.2 Similarity Method

Another heuristic in identifying the BMU may be to compile a list of terms, or dimensions, similar to both SOMs. Then, for each term, find the node with the heaviest weight of that specific term in both the source and destination SOM. Once identified, create a connection between the BMUs. One may also incorporate thresholding; any BMU in the receptive SOM that does not exceed a specified threshold is barred from creating a connection.

It is noted that this method is similar to the Euclidean method; however, in the Euclidean method, the entire input lattice is used to compute the similarity between

SOMs. In the similarity method, just one dimension of the input lattice is used in determining the BMU.

7.5.3 Clustering Method

The final method of determining the BMU and neighborhood is a hybridized version of the two, previously described methods and clustering. The clustering method is a technique for combating redundant information that is located in separate SOMs. Briefly, the method creates connections utilizing one of the formerly mentioned methods. Once connections are made, the Clustering Method clusters, or groups, similar connections in both the target SOM as well as the original SOM. If a majority of the members in one cluster are transformed, or mapped, to another cluster on the other SOM, it implies that information in one SOM may be redundant, hence, the duplication. As such, the neighborhood of each node in the SOM is dependent upon the membership of the clusters, and the neighborhood radius of the connection is enlarged if needed; this ensures further exploration of the connection space [Figure 7.6].

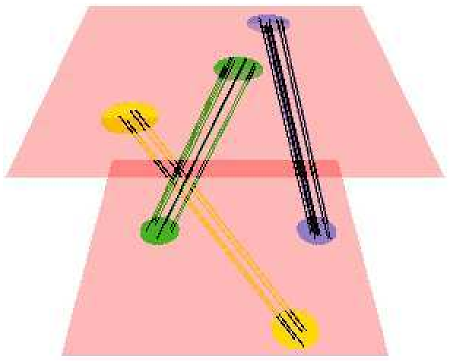


FIGURE 7.6: Grouping Connections

Many methods of clustering exist and many may be applied to PSOM connection theory [11, 13, 14]. Clustering, in general, is an unsupervised learning process, and many algorithms for clustering exist. Furthermore, clustering, much like the SOM algorithm, is the process of identifying variable features among elements and categorizing them according to their similar or dissimilar features, creating sets with agreeable characteristics.

Of all the clustering algorithms, perhaps the most popular and most simplistic is the K-Means algorithm. The K-Means algorithm begins by assigning k centroids in the space. These centroids, or centers of gravity, can be assigned randomly or algorithmically. Once the centroids are defined, each element in the space is then paired with a centroid of closest proximity, using various distance metrics. Once all elements have been matched, the algorithm recomputes the centroids of the space, and elements are regrouped. One will notice the centroids tend to drift towards a local extreme. This process of computing the centroids and regrouping elements is iterated until the system eventually converges [14]. Once converged, the elements are completely grouped. The equation for this iterative process is presented below.

$$V=\sum_{i=1}^{k}\sum_{j=1}^{n}\|x_j-c_k\|^2 ,$$

where: V = the variance,
k = the amount of clusters, and
j = the node of interest.

From observation, the K-Means Algorithm is minimizing the error of the function, or the objective function, which is the distance metric. Although relatively fast, the effectiveness of K-Means algorithm is disputable. First, the number of clusters, which is the number of centroids, is a priori knowledge and must be ascertained prior to runtime [13]. Second, the algorithm is sensitive to the initial state of each of the centroids. As previously mentioned, the algorithm converges on a local extreme, which does not indicate that the convergence is the global extreme. Therefore, many iterations of the algorithm must be performed, each with new initial positions of the centroids, choosing the resultant that furthest reduced the objective function. Finally, the K-Means Algorithm is considered as hard-clustering, for each element in the set must be assigned a group although it may have no relationship to a neighbor in the group.

Although it can be implemented, the K-Means Algorithm lacks the robustness and reliability desired of the system. Without further adaptation and modification of the algorithm, the algorithm is inefficient and bland. Consequently, a more diverse and powerful algorithm for clustering exists; it is the EM Clustering Algorithm, which is a model-based algorithm [11, 13, 14, 30].

EM Clustering, as opposed to K-Means Clustering, is soft clustering. In soft clustering, for each cluster in the set {1, …, k}, a probability of membership is assigned to each document {1, …, i}, where the sum of probabilities is equal to one.

$$\sum_{c=1}^{k} q_{i,c} = 1 \ ,$$

where: $q_{i,c}$ = the probability of document *i* in cluster *c*.

To perform EM Clustering, one must first select a cluster $c \in \{1, \ldots, k\}$, assigning each cluster a specific probability μ_c, where the summation of probabilities is equal to one. Once selected, for each document in the data set, one computes the probability of that element given the model parameter for the specific cluster, or θ_c. The model parameter for the system, in this instance, is the Euclidean distance calculated from the centroid, or center of gravity. At the initial epoch, this centroid is surmised from the given data set. As the data generated is a mixture model, the probability of the document is computed as follows…

$$p(x)=\sum_{c=1}^{k}\mu_c p_c(x|\ \ _c)$$

where,

$$p_c(x|\ \ _c)=\frac{p(\ \ _c|x)\,p(x)}{p(\ \ _c)};\ \ p(\ \ _c|x)=\frac{p(x\cap\ \ _c)}{p(x)}\ ,$$

or, when one assumes the mixture model to be Gaussian, one assumes…

$$p_c(x|\ \ _c)\propto\exp\left(-\frac{(\ \ _c-x)^2}{2\ \ ^2}\right)\quad (1).$$

Given the probability of x, or the document, one may then compute the probability of the document from a given cluster, which is the posteriori…

$$p(y|x)=\frac{\mu_c p_y(x|\ \ _y)}{\sum_{c=1}^{k}\mu_c p_c(x|\ \ _c)}\quad (2).$$

It is known for each cluster, or label, there exists a hidden variable, q_y. This hidden variable is a probability measure; in truth, this hidden variable completes the data set when combined with the observed data. It is also known that a mutual entropy exists between the q_y and p(y|x), which is defined as…

$$\sum_{c=1}^{k}q_c*\ln\frac{q_c}{p(c|x)}$$

It is apparent that this entropy is never negative – due to the natural log – and that no entropy may be achieved when q_c is equal to p(c|x). Knowing this, one substitutes this expression into (2), resulting in…

$$\ln\sum_{c=1}^{k}\mu_c p_c(x|\ \ _c)\geq\sum_{c=1}^{k}q_c\ln\frac{\mu_c p_c(x|\ \ _c)}{q_c}\quad(3),$$

where it is known the maximum value is obtained when q_y = p(y|x), which is defined in (2).

Now, to apply maximum-likelihood estimation, which selects a parameter to maximize the observed data's likelihood: $\prod_{i=1}^{n}p(x_i)$ for each element in the input vector; in essence, one is solving for θ_c.

$$\max_{\ _1,\ldots,\ _k}\sum_{i=1}^{n}\ln\sum_{c=1}^{k}\mu_c p_c(x_i|\ \ _c)\quad(4).$$

Doing a simple substitution of (3) into (4), yields the result…

$$\max_{\ _1,\ldots,\ _k}\sum_{i=1}^{n}\max_{q_{i,1},\ldots,q_{i,k}}\sum_{c=1}^{k}q_{i,c}\ln\frac{\mu_c p_c(x_i|\ \ _c)}{q_{i,c}}.$$

In short, in the E-step, one makes θ_c static, solving for $q_{i,c}$. Once solved, in the M-step, one makes $q_{i,c}$ static, solving for θ_c. Through constant iterations, the system will converge on a local optimum, and clusters will form from disorder.

One may apply a Gaussian function to EM clustering. For $p_c(x_i \mid \theta_c)$ one substitutes a Gaussian function as defined in (1). Now to solve for the max, one simply takes the derivative with respect to θ_c, equating the expression to zero and solving for θ_c. Doing so will result in the following equations…

$$q_{i,c} = \frac{\exp\left(-\frac{\left(x_i - {}_c\right)^2}{2\ {}^2}\right)}{\sum_{l=1}^{k} \exp\left(-\frac{\left(x_i - {}_l\right)^2}{2\ {}^2}\right)},$$

$${}_c = \frac{1}{\sum_{i=1}^{n} q_{i,c}} \sum_{i=1}^{n} q_{i,c}\, x_i .$$

However, EM model, too, is inefficient due to the fact that the number of clusters must be known a priori. As the PSOM can be relatively large and broad, requiring user intervention – determining the number of clusters – is inadequate. The number of clusters must be computed during runtime; the number of clusters must be known without supervision.

Cross-validation, very similar to genetic algorithms, provides effective methods for estimating the amount of clusters given a particular set. Of the many cross-validation methods, V-Fold Cross-Validation provides a constructive approximation as to the number of clusters.

In V-Fold Cross-Validation one first divides the sample into k-partitions {P_1, P_2, …, $P_{k\}}$. Each partition represents a unique cluster in the set. To begin optimizing the cluster number, select one partition at random, P_r, from the partitioned set. This random partition shall be used for test verification and validation. Consequently, for each k in the partition set, one trains the set using k – 1 partitions, omitting the partition P_r. Once training is complete, validate the algorithm using the omitted partition, P_r. In validating the algorithm, the following metric is used…

$$Error = \frac{1}{k} \sum_{i=1}^{k} err\left(P, P_i\right),$$

where: P = the set of partitions excluding P_i.

To determine an optimal cluster number, reiterate this method of cross-validation process for new values of k. The partition set with minimal error corresponds to the most advantageous cluster number.

To reiterate, each terminal of the connection has an associated neighborhood, and each of the neighborhoods has a boundary. One has the important task in determining the extents of the neighborhoods. Therefore, it is accurate when describing each cluster to reference a distance metric, which properly defines these boundaries of the clusters.

CHAPTER 8 – RESULTS FROM IMPLEMENTING A PRISMAL SELF-ORGANIZING MAP

In creating the PSOM to solve the SSD paradigm, a corpus of documents were divided into various random sets. These sets are analogous to different media types, containing textual data files. Once divided, the preprocessing phase commences and reduces the dimensionality of the dataset and identifies relevant terms. Afterward, a map is created, and each node in the map is fully connected to that of the created input vector. Initializing the map, one uses random weights, connecting the nodes of the map to the nodes of the input vector. Finally, during each epoch, the input vector is fed each document's relevant variables and a corresponding BMU is identified along with its neighbors and is rewarded accordingly. A predefined amount of iterations were performed, and a SOM was created [Figure 8.1].

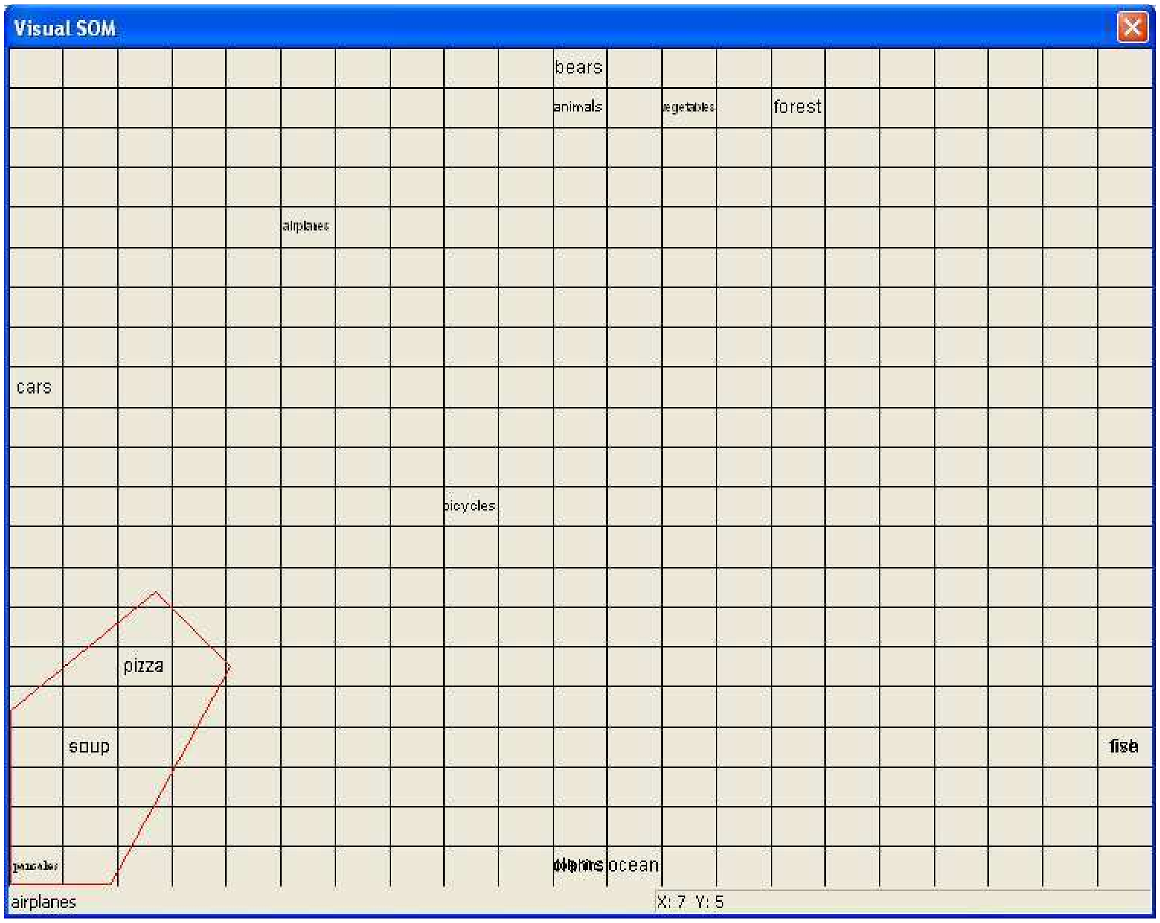


FIGURE 8.1: A Visual Representation of a Low-Dimensional SOM[3]

The SOMs from each set were then assembled, connections were created, and the result was a fully connected PSOM.

The PSOM was tested extensively using the Cranfield corpus – which is a collection of documents pertaining to the aerospace industry, including aerodynamics, dynamics, statics, fluid mechanics, et cetera – and a corpus of smaller, more common documents. For clarity the Cranfield corpus is composed of 1400 documents; on the other hand, the other corpus utilized consists of only 200 textual documents.

[3] Notice the clusters. In one example – the cluster in red – pizza, soup, and pancakes create a close cluster as they are all foods.

Radial neighborhoods in the PSOM initially proved not to be advantageous. In a few experiments, the boundaries between clusters created by the SOM are extremely sharp. By sharp, it is understood there is a definitive boundary between clusters, and there is a drastic change in the dimensions at these boundaries. To alleviate this deficiency, the size of the map was increased two twenty times the size of the input lattice, which relaxed the boundaries. On the other hand, irregular neighborhoods were not prone to this discrepancy; therefore, further testing was only performed on irregular neighborhoods even though the map extensions were still enlarged.

Queries are performed by inputting a keyword into the system. The system proceeds to identify nodes with high fitness associated with the keyword for each individual SOM. Once identified, the SOM returns a list of relevant terms within proximity of the neighborhood [Figure 8.2].

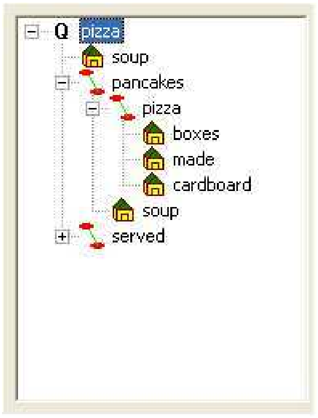


FIGURE 8.2: Relevant Terms Returned from a Query

If the term is associated with a connection, it is indicated in the user interface and the connection is traversed and then perused for associated terms. Also, if a document is associated with a particular node, the document's name, location, and media name are prominently displayed in the user interface.

Information retrieval evaluation is measured by two metrics: precision and recall [3, 9]. Precision is a unitless metric, more precisely a percentage, which identifies the proportion of relevant documents returned by the query. It is defined as…

$$P = \frac{D_r}{D_t},$$

where: D_r = the number of relevant documents retrieved, and
D_t = the total number of documents retrieved.

Recall, very similar to precision, identifies the proportion of relevant documents returned by the query with respect *to the total number of relevant documents*. It is defined as…

$$R = \frac{D_r}{N_r},$$

where: D_r = the number of relevant document retrieved, and
N_r = the total number of relevant documents in the corpus satisfying the request.

One must note that precision and recall are subjective values. Consider, for instance, two identical queries. As they are identical, they produce equivalent results; however, the relevance of document is subjective. One user may deem a resultant document useful while another user may estimate the document as having little intrinsic value; the relevance is subjective to the user.

In testing the PSOM, a corpus was randomly divided into various sets. It is important to note that these sets were, in fact, random and the location of each file is unbeknownst to the analyst. From the sets, SOMs were created, and then a PSOM constructed. Given a subject, the analyst was asked to utilize the PSOM to determine files of interest, including their locations.

Not surprisingly all methods had very high precisions: 83 to 100 percent. Even the recalls were high: 75 to 100 percent. These high rates are attributed to the SOMs ability to offer clarity and advice of further terms to append to the query, for terms of interest are within close proximity of the BMU associated with the query. The PSOM evolves this idea even further by offering more related terms through correlated connections in additional SOMs.

It is also noted that the clustering method was only tested in SOMs with large map extents, which implies high-dimensionality. It was perceived that the clustering method would perform rather poorly utilizing small maps, and initial testing proved this, for few connections were made but were still grouped. As many of the connections were grouped, the neighborhood of the connected SOM was fairly large and included many false neighbors and terms. In a large map set, including partial duplicates of documents in SOMs, the clustering method encouraged exploration, for the neighborhood radius of the terminals were enlarged.

Calculating precision and recall in a PSOM is quite difficult. As the query monotonically grows, the focus of the query drifts, for new areas of interest are found. However, the power of the PSOM is not to define relevance; it is the ability to recall information through stimulation. Consider, for example, an individual desires a document pertaining to “The Bellagio Casino,” but all he can recall is what he’s looking for is in “Nevada.” He starts the query. Possible neighbors include “Sierra,” “Tahoe,” and “Vegas.” In perusing the returned list of neighboring terms, he recalls the item he’s seeking is related to “Vegas.” He expands his query; the results returned are: “city,” “hot,” and “casino.” He is beginning to recall the item he’s looking for! He delves deeper into his query by expanding on the term “casino.” The results now include

“Hilton,” “Wynn,” and “Bellagio.” Voila! He now remembers! “Bellagio” ultimately returns a list of documents – as well as their media locations – pertaining to the Bellagio Casino in Las Vegas, Nevada. It is no wonder the SOM as well as the PSOM provide a procedure for information retrieval and recollection.

CHAPTER 9 – Conclusion

The PSOM offers an ample solution to solving the segmented, static database, or SSD. By finding correlations amongst nodes in various SOMs – that are in series – the PSOM easily traverses maps, broadening exploration and retrieval. In doing so, the PSOM offers reliable and accurate queries, reducing repetitious tasks such as mounting, searching, and unmounting media, endeavoring to gain insight and knowledge in a timely manor. And, although the scope of the thesis was limited to that of segmented, static databases, the methods presented here may be applied to other areas of study should the circumstances merit a PSOM. It is a powerful tool capable of exploring a multidimensional space. The PSOM truly is an amazing phenomenon.

CHAPTER 10 – FUTURE WORK

As the basis for this thesis was to solve the statically, segmented database paradigm, the dimensionality of each individual SOM and PSOM is extremely large. Therefore, when creating connections between the elements in the PSOM, the elements must be syntactically similar. If the dimensionality of the elements were reduced, or a commonality existed between the SOMs, one could conceivably reference a data structure, perhaps a taxonomy structure, to find similarities and dissimilarities – much like a membership function pertaining to Fuzzy Logic – between the elements.

Therefore, when creating connections, an element could – conceivably – have more than one connection, and the weights associated with each connection are dependent on the resultant data structure. For instance, suppose one creates a PSOM and restricts the boundary of the PSOM to the animal kingdom. Referencing a taxonomy structure, the PSOM notes that the fuzzy relation between a lion and a tiger is considerably strong; the PSOM creates a fuzzy connection. Also, the PSOM notes another fuzzy relation exists among a lion and the domestic cat; the PSOM, again, creates another fuzzy connection. Connections – as previously described in this work – are discrete; they either exist or do not exist, and they do not have differing weights among connections. Fuzzy connections, on the other hand, have varying weights. As depicted in the previous example, a lion is more similar to a tiger, but it still has a relationship with a domestic cat. Because the relationship is strong with the tiger, the connection has a higher weight than that of the connection with the domestic cat.

Other clustering algorithms exist. In fact, another popular clustering algorithm may be implemented in parallel to the taxonomy. The Fuzzy C-Means Algorithm is another unsupervised clustering algorithm. The algorithm, in short, allows data elements to reside, or partake in membership, in more than one cluster. Therefore, if using this algorithm, terminals of connections may reference more than one neighborhood in the underlying SOM, much like EM Clustering.

Finally, as the system is query-driven, relevance feedback should be included in the system. Relevance feedback is a measure of ranking returned documents to a particular query [8, 33]. Qualitatively, it is the process of identifying documents that are relevant or unrelated to a specific query. Once ranked, the documents modify subsequent queries, creating a closed-loop system. Implementing a ranking, or closed-loop, system enhances recall and precision, which are quantitative.

REFERENCES

[1] "Kohonen's Self Organizing Feature Maps." *AI-Junkie*. http://www.ai-junkie.com/ann/som/som1.html

[2] Atkinson-Abutridy, John, Chris Mellish, and Stuart Aitken. "Combining Information Extraction with Genetic Algorithms for Text Mining." *IEEE Intelligent Systems*. Vol 19-3. May – June 2004. pp. 22-30.

[3] Berry, Michael W., and Murray Browne. "Understanding Search Engines: Mathematical Modeling and Text Retrieval." 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2005.

[4] Bryson, Bill. "The Mother Tongue: English and How It Got That Way." New York: Avon Books, 1991.

[5] Buntine, Wray, and Aleks Jakulin. "Applying Discrete PCA in Data Analysis." *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*. Vol. 70. 2004. pp. 59-66.

[6] Butz, M.V., et al. "Toward a Theory of Generalization and Learning in XCS." *IEEE Transactions on Evolutionary Computation*. Vol 8-1. Feb. 2004. pp. 28-46.

[7] Chen, Guanrong, and Trung Tat Pham. "Introduction to Fuzzy Systems." Boca Raton, FL: Chapman and Hall, 2006.

[8] Crestani, Fabio. "Comparing Neural and Probabilistic Relevance Feedback in an Interactive Information Retrieval System." *1994 IEEE World Congress on Computational Intelligence*. Vol. 5, 27 June – 2 July 1994. pp. 3426 – 30.

[9] ---. "Learning Strategies for an Adaptive Information Retrieval System using Neural Networks." *1993 IEEE International Conference on Neural Networks*. Vol. 1, 28 Mar. – 1 Apr. 1993. pp. 244 – 9.

[10] Dam, Hai H., Hussein A. Abbass, and Chris Lokan. "Learning Classifier Systems and Other Genetics-Based Machine Learning: DXCS: An XCS System for Ditributed Data Mining." *Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*. 2005. pp 1883-90.

[11] Everitt, Brian. "Cluster Analysis." London: Halsted Press, 1980.

[12] Goldberg, David E. "Genetic Algorithms: In Search, Optimization, and Machine Learning." Reading, MA: Addison-Wesley, 1989.

[13] Hair, Joseph F., Jr., Rolph E. Anderson, and Ronald L. Tatham. "Multivariate Data Analysis with Readings." New York: Macmillan, 1987.

[14] Hartigan, John A. "Clustering Algorithms." New York: Wiley, 1975.

[15] Honkela, Timo, et al. "Exploration of Full-Text Databases with Self-Organizing Maps." *1996 IEEE International Conference on Neural Networks*. Vol. 1, 3-6 June 1996. pp. 56-61.

[16] ---. "Self-Organizing Maps in Natural Language Processing." Diss. Helsinki University of Technology, 1997. http://www.cis.hut.fi/~tho/thesis/

[17] Kohonen, Teuvo. "Self-Organizing Maps." 3rd ed. New York: Springer, 2001.

[18] ---. "The Self-Organizing Map." Ed. V. Rao Vemuri. *Artifical Neural Networks: Concepts and Control Applications*. Los Alamitos, CA: IEEE Computer Society Press, 1992.

[19] ---, et al. "Self Organization of a Massive Document Collection." *IEEE Transactions on Neural Networks*. Vol. 11.3, May 2000. pp. 574-85.

[20] ---, et al. "Self-Organizing Maps of Massive Document Collections." *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks*. Vol. 2, July 2000. pp. 3-9.

[21] Kwok, K.L. "A Neural Network for Probabilistic Information Retrieval." *Proceedings of the 12th Annual International ACM SIGR Conference on Research and Development in Information Retrieval*. 1989. pp. 21-30.

[22] Mital, V., and T.D. Gedeon. "A Neural Network Integrated With Hypertext for Legal Document Assembly." *1992 Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*. Vol. 2, 7-10 Jan. 1992. pp. 533 –9.

[23] Mohammadian, Masoud. "Intelligent Agents for Data Mining and Information Retrieval." Hershey, PA: Idea Group, 2004.

[24] Robertson, Stephen. "Understanding Inverse Document Frequency: On Theoretical Arguments for IDF." *Journal of Documentation 60*. Vol. 5. pp. 503-20.

[25] Salton, Gerard. "Automatic Information Organization and Retrieval." New York: McGraw-Hill, 1968.

[26] ---. "A Theory of Indexing." Philadelphia: Society for Industrial and Applied Mathematics, 1975.

[27] Sy, Bon K., and Arjun K. Gupta. "Information-Statistical Data Mining: Warehouse Integration with Examples of Oracle Basics." Boston: Kluwer Academic, 2004.

[28] Symeonidis, Andreas L., and Paricles A. Mitkas. "Agent Intelligence Through Data Mining." New York: Springer, 2005.

[29] Tambe, Sanjeev S, Bhaskar D. Kulkarni, and Pradeep B. Deshpande. "Elements of Artificial Neural Networks with Selected Applications in Chemical Engineering and Chemical and Biological Sciences." Louisville, KY: Simulation and Advanced Controls, 1996.

[30] Weis, Sholom M., et al. "Text Mining: Predictive Methods for Analyzing Unstructured Information." New York: Springer, 2005.

[31] Wong, S.K.M., Y.J. Cai, and Y.Y. Yao. "An Application of Neural Networks in Adaptive Information Retrieval." *1993 IEEE International Conference on Neural Networks*. Vol. 1, 28 Mar. – 1 Apr. 1993. pp. 1667 – 71.

[32] Yen, John, and Reza Langari. "Fuzzy Logic: Intelligence, Control, and Information." New York: Prentice Hall, 1998.

[33] Yu, C.T., W.S. Luk, and T.Y. Cheung. "A Statistical Model for Relevance Feedback in Information Retrieval." *Journal of the ACM*. Vol. 23-2. April 2006. pp. 273-86.