Suffix Tree Based Message Passing Algorithm

Neha Saxena¹, and Shri Shailendra Shrivastava²
¹ Samrat Ashok Technological Institute, dept(IT), Vidisha(M.P), India
   Email: nehasaxena.2830@gmail.com
² Samrat Ashok Technological Institute, dept (IT), Vidisha(M.P), India
   Email: shailendrashrivastava@rediffmail.com

Abstract—In Document Clustering, Phrase is very effective as compared to the clustering of words taken from the text. Here Suffix Tree Document (STD) model is used for the extraction of phrases from the text. Then weights of the phrases are found out through tf-idf weighting scheme to result in unique feature term in the Vector Space Document (VSD) model, document similarity is found out with the help of phrases. Phrase-based document similarities are applied to the message passing algorithm and develop a new document clustering approach. Our objective is to cluster a collection of text based on phrases so as to improve the speed and performance of clustering. The Performance of clustering algorithm is measured by F-measure, entropy and purity.

Index Terms—Document clustering, suffix trees, message passing algorithm, F-measure, Purity, entropy

I. INTRODUCTION

The amount of available information on the Web is increasing rapidly. The publicly indexable Web contains an estimated 800 million pages as of February 1999, encompassing about 15 terabytes of information or about 6 terabytes of text after removing HTML tags, comments, and extra whitespace. “The revolution that the Web has brought to information access is not so much due to the availability of information (huge amounts of information has long been available in libraries and elsewhere), but rather the increased efficiency of accessing information, which can make previously impractical tasks practical”. But how can users find the information they are seeking in such an unorganized, unstructured and decentralized place?

Data mining is the process of discovering new patterns from large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics and database systems. The overall goal of the data mining process is to extract knowledge from a data set in a human-understandable structure. Data mining and machine learning technologies have already achieved significant success in many knowledge engineering areas including classification, regression and clustering algorithm. Document clustering has long been studied as a postretrieval document visualization technique to provide an intuitive navigation and browsing mechanism by organizing documents into groups, where each group represents a different topic [1], [2]. In general, the clustering techniques are based on four concepts: data representation model, similarity measure, clustering model, and clustering algorithm. Most of the current documents clustering methods are based on the Vector Space Document (VSD) model [3]. The common framework of this data model starts with representation of any document as a feature vector of the words that appear in the documents of a data set. A distinct word appearing in the documents is usually considered to be an atomic feature term in the VSD model, because words are the basic units in most natural languages (including English) to represent...
semantic concepts. In particular, the term weights (usually tf-idf, term-frequencies and inverse document-frequencies) of the words are also contained in each feature vector. The similarity between two documents is computed with one of the several similarity measures based on the two corresponding feature vectors, e.g., cosine measure, Jaccard measure, and euclidean distance. To achieve a more accurate document clustering, a more informative feature term—phrase—has been considered in recent research work and literature. A phrase of a document is an ordered sequence of one or more words. The Suffix Tree Document (STD) model was proposed by Zamir et al. [4] and Zamir and Etzioni [5]. The STD model considers a document as a sequence of words, not characters. A document is represented by a set of suffix substrings, the common prefixes of the substrings are selected as phrases to label the edges (or nodes) of a suffix tree. Whereas, Suffix tree is a data structure that admits efficient string matching and querying. It has been studied and used extensively in fundamental string problems and applications such as large volumes of biological sequence data searching, approximate string matches, and text features extraction in spam e-mail classification. In a similar manner, a suffix tree of a set of strings, called a generalized suffix tree, is a compact trie of all the suffixes of all the strings in the set. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of a string in S.

The rest of the paper is organized as follows: Section 2 discusses the related work. Section 3 illustrates the brief review of proposed work i.e., suffix trees, phrase-based document similarity and message passing algorithm. Section 4 illustrates the Result Analysis with the help of performance metrics or parameters used to check the effectiveness and efficiency evaluation for the phrase based document similarity and message passing clustering algorithm.

II. RELATED WORK

Text document clustering has been traditionally investigated as a means of improving the performance of search engines by preclustering the entire corpus [6], and a postretrieval document browsing technique as well [1], [2]. Hierarchical Agglomerative Clustering (HAC) algorithm might be the most commonly used algorithm among numerous document clustering algorithms. Generally, there are three variants from this algorithm: single-link, complete-link, and group-average. In practice, the HAC algorithm can often generate high-quality clusters with a tradeoff of the higher computation complexity [7]. K-Nearest Neighbor (K-NN) algorithm is well known for classification. It has also used for document clustering [8], [9]. In the traditional document models such as the VSD model, words or characters are considered to be the basic terms in statistical feature analysis and extraction. To achieve a more accurate document clustering, developing more informative features has become more and more important in information retrieval literature recently. Suffix tree is a data structure that admits efficient string matching and querying. It has been studied and used extensively in fundamental string problems and applications such as large volumes of biological sequence data searching [10], approximate string matches [11], and text features extraction in spam e-mail classification [12]. The STD model was first proposed in 1997 [3], [4]. Different from document models which treat a document as a set of words and ignore the sequence order of the words [13], the STD model considers a document to be a set of suffix substrings, and the common prefixes of the suffix substrings are selected as the phrases to label the edges of the suffix tree. The STD model considers a document d as a string consisting of words w1 w2 . . . w m, not characters. The suffix tree of a document d is a compact trie containing all suffix substrings of the document d. There are three kinds of nodes in the suffix tree: the root node, internal nodes, and leaf nodes. Each internal node at least has two children. Each edge is labeled with a nonempty substring of a document called a phrase. Then, each leaf node in the suffix designates a suffix substring of a document; each internal node represents a common phrase shared by at least two suffix substrings. The similarity of two documents is defined as the more internal nodes shared by the two, the more similar the documents tend to be.

III. PROPOSED WORK

A. Document Pre-processing

Before the document clustering, a document “cleaning” procedure is executed for all documents in the data sets:

- First, all non-word tokens are stripped off.
- Second, the text is into words.
• Third, all stop words are identified.
• Since the length of a word is variable, it is quite difficult to implement a suffix tree based on words directly. To solve the problem, we build a wordlist to store all keywords in alphabetical order.
• In the wordlist, a unique integer number (called a word_id) is assigned to each keyword so that we can use the word_id to replace the corresponding word in the “cleaned” document.
• Finally, each document becomes an array of word_id for the suffix tree construction.

B. Standard Suffix Tree Document Model
The STD model considers a document d as a string consisting of words $w_1w_2 \ldots w_m$, not characters. The suffix tree of a document d is a compact trie containing all suffix substrings of the document d. In a similar manner, a suffix tree of a set of strings, called a generalized suffix tree, is a compact trie of all the suffixes of all the strings in the set[16]. Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty sub-string of words of a string in $S$. Fig. 1 is an example of a suffix tree composed from three documents “cat ate cheese,” “mouse ate Cheese too,” and “cat ate mouse too.”

![Suffix Tree Diagram](image_url)

In particular, there exists some leaf nodes labeled with an empty phrase; they are generated by Ukkonen’s algorithm which is used to build suffix tree, and denote the end of the corresponding documents. We call these leaf nodes “terminal nodes” in our work. With the exception of the terminal nodes and the root node, each node in the suffix tree, either an internal node or a leaf node represents a nonempty phrase that appears in at least one document in the data set. The same phrase might occur in different edges of the suffix tree. For instance, there are three different edges labeled with the same phrase of “cheese” in the suffix tree of Fig. 1. The table below shows the phrase extracted by suffix tree from three documents.

<table>
<thead>
<tr>
<th>Node</th>
<th>Phrase</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>cat ate</td>
<td>1,3</td>
</tr>
<tr>
<td>b</td>
<td>ate</td>
<td>1,2,3</td>
</tr>
<tr>
<td>c</td>
<td>cheese</td>
<td>1,2</td>
</tr>
<tr>
<td>d</td>
<td>mouse</td>
<td>2,3</td>
</tr>
<tr>
<td>e</td>
<td>too</td>
<td>2,3</td>
</tr>
<tr>
<td>f</td>
<td>ate cheese</td>
<td>1,2</td>
</tr>
</tbody>
</table>

TABLE 1: SHOWING THE EXTRACTED PHRASES FROM THE SUFFIX TREE.
Several linear time algorithms for constructing suffix trees exist\cite{14}\cite{15}. To be precise, these algorithms also exhibit a time dependency on the size of the vocabulary (or the alphabet when dealing with character based trees): they actually have a time bound of $O(m \cdot \min(\log |L|, \log m))$ where $m$ is the string’s length and $|L|$ is the size of the language.

Weiner’s algorithm
McCreight’s algorithm
Ukkonen’s algorithm (here it is used in proposed work)

Ukkonen’s algorithm, however, is simpler and has a potentially useful online characteristic\cite{14}. While McCreight’s algorithm requires the whole string to be present before starting, Ukkonen’s method can process each character in order. Ukkonen’s algorithm constructs a series of implicit suffix trees, the last of which is converted into a true suffix tree. An implicit suffix tree is a simplified suffix tree without the addition of a terminating character to the string and without inserting suffixes that match the prefix of other suffixes fig 2.

![Fig2: Suffix tree for string xabxa$](image)

C. Phrase-based document similarity

By mapping each node $v$ labeled by a nonempty phrase into a feature of $M$-dimensional term space, each document $d$ can be represented as a feature vector of the weights of $M$ node terms in the VSD model. It is very easy to understand that the document frequency of each node $df(v)$ is the number of the different documents that have traversed node $v$; the term frequency $tf(v, d)$ of a node $v$ with respect to document $d$ is the total traversed times of the document $d$ through node $v$.

Term frequency – inverse document frequency (tf-idf) is commonly used IR technique for assigning weights to individual words (terms)\cite{17}\cite{18}. We calculate the tf-idf $(wi, d)$ score of word $wi$ in document $d$ using the following formula:

$$tf-idf(wi, d) = (1 + \log(tf(wi, d))) \cdot \log(1 + \frac{N}{df(wi)})$$

where $tf(wi, d)$ is the number of occurrences of word $wi$ in document $d$, $N$ is the total number of documents in our document set and $df(wi)$ is the number of documents that term $wi$ appears in.

After obtaining the term weights of all nodes, it is easy to apply traditional similarity measures such as the cosine similarity to compute the similarity of any two documents.

Here, the cosine similarity measure is used to compute the pair-wise similarities of all documents.

Let vectors $\vec{x}_d = [x_1, x_2, x_3, \ldots, x_m]$ and $\vec{y}_d = [y_1, y_2, y_3, \ldots, y_n]$ denote two documents $d_x$ and $d_y$, where $x$ are the weights of corresponding node term $vi$, respectively. Then, the similarity of two documents is calculated by Eq.(1)

$$sim_{xy} = \frac{\sum_{i=1}^{M} x_i y_i}{\sqrt{\sum_{i=1}^{M} x_i^2 \sum_{i=1}^{M} y_i^2}}$$
D. Message passing algorithm

Message Passing simultaneously considers all data points as possible exemplars, exchanging real-valued messages between them until a high quality set of exemplars (and corresponding clusters) emerges. Messages are updated on the basis of simple formulae that reflect sum-product or max-Product update rules and, at any point in time, the magnitude in each message reflects the current affinity that one point has for choosing another data point as its exemplar, hence the name “Message Passing”. Message Passing takes as input a collection of real-valued similarities between data points, \( \{s(i, k)\} \), where each similarity \( s(i, k) \) indicates how well the data point with index \( k \) is suited to be the exemplar for data point \( i \)\[19\][20]. In general the algorithm works in three steps:

a) Update responsibilities given availabilities. Initially this is a data driven update, and over time lets candidate exemplars competition for ownership of the data.

b) Update availabilities given the responsibilities. This gathers evidence from data points as to whether a candidate exemplar is a good exemplar.

c) Monitor exemplar decisions by combining availabilities and responsibilities. Terminate if reach a stopping point (e.g. insufficient Change). The update rules require simple local computations and messages are exchanged between pairs of points with known similarities.

- Responsibility \( r(i, k) \): accumulated evidence of data \( k \) to be the exemplar for data \( i \) as shown in fig 3(a)
- Availability \( a(i, k) \) : accumulated evidence of data \( i \) pick data \( k \) as the exemplar as shown in fig 3(b)

![Fig.3 (a) Responsibilities Messages](image)

![Fig.3 (b) Availabilities Messages](image)
When the goal is to minimize squared error, each similarity is set to a negative squared error (Euclidean distance):

$$s(i,k) = -\|x_i - x_k\|^2$$  \hspace{1cm} (2)

The preference of point $i$, called $p(i)$ or $s(i,i)$, is the a priori suitability of point $i$ to serve as an exemplar. Preferences can be set to a global (shared) value, or customized for particular data points. High values of the preferences will cause Message Passing to find many exemplars (clusters), while low values will lead to a small number of exemplars (clusters).

**E. Steps of proposed Algorithm**:

**Input:** Similarity Matrix and Preference;

**Output:** Clustering Solution;

**Step 1:** Construct the similarity Matrix based on the cosine distance for finding similarity of feature vectors of text.

**Step 2:** Calculate the Preference.

$$S(l, l) = \sum_{k \neq l}^N s(l, k) \frac{1}{N-1} \quad 1 \leq l \leq N$$  \hspace{1cm} (3)

**Step 3:** The responsibility message $r(i, k)$ sent from data point $i$ to $k$ is to serve as the exemplar for the data point $i$.

- Responsibilities are calculated as in Eq.(4) follows:

$$r(i,k) \leftarrow s(i,k) - \max_{k' \neq k} \{ a(i,k) + s(i,k') \}$$  \hspace{1cm} (4)

In the first iteration, because the availabilities are zero, $r(i,k)$ is set to the input similarity between point $i$ and point $k$ as its exemplar, minus the largest of the similarities between point $i$ and other candidate exemplars. The responsibility $r(k,k)$ “self-responsibility” is set to the input preference that point $k$ be chosen as an exemplar, $s(k,k)$, minus the largest of the similarities between point $i$ and all other candidate exemplars.

**Step 4:** The availability message $a(i,k)$ sent from candidate exemplar point $k$ to $i$ reflects the accumulated evidence for how appropriate it would be for point $i$ to choose point $k$ as its exemplar.

- Availabilities are calculated in Eq.(5) as follows:

$$a(i,k) \leftarrow \min \{ 0, r(k,k) + \sum_{i' \neq k} \max \{ 0, r(i',k) \} \}$$  \hspace{1cm} (5)

The “self-availability” $a(k,k)$ is updates as

$$a(k,k) \leftarrow \sum_{i' \neq k} \max \{ 0, r(i',k) \}$$

This message reflects accumulated evidence that point $k$ is an exemplar, based on the positive responsibilities sent to candidate exemplar $k$ from other points.

**Step 5:** Set damping factor, convergence iteration and maximum iteration. Damping factor ($\text{dampfact}$): When updating the messages, it is important that they be damped to avoid numerical oscillations that arise in some circumstances.

$$\text{msgnew} = (\text{dampfact})(\text{msgold}) + (1 - \text{dampfact})(\text{msgnew})$$

**convits and maxits:** Affinity propagation iteratively computes responsibilities and availabilities. The algorithm terminates if decisions for the exemplars and the cluster boundaries are unchanged for convits iterations, or if maxits iterations are reached.

**Step 6:** At any point during proposed algorithm availabilities and responsibilities can be combined to identify exemplars. For point $i$, the value of $k$ that maximizes $a(i,k) + r(i,k)$ either identifies point $i$ as an exemplar if $k = i$, or identifies the data point that is the exemplar for point $i$.

**Step 7:** Repeat the process from step 3 to step 7.

The algorithms is halted after a fixed number of iterations or after the exemplars do not change for a given number of iterations.

**Step 8:** Stop.

**IV. RESULT ANALYSIS**

To measure these performance parameters datasets of three known classes from Ohsumed medical dataset[21] is used, each containing 10 documents results in dataset of 30 documents. The proposed algorithm clusters large dataset with great accuracy. The main purpose of the proposed algorithm is to improve accuracy and execution time.
Performance Parameters: We measure the performance of our algorithm in the form of following parameters:

(a) F-Measure: F-measure is a measure of a test’s accuracy. Recalling C={C1,C2,C3,…..Ck} is a clustering of data set D of N documents, let C*={C*1 ,C*2 ……C*l } designate the “correct” class set of D. Then, the recall of cluster j with respect to class i, as rec(i,j) is . The precision of cluster j with respect to class i,prec(i,j) is defined . F-Measure combines both values according to the following formula combines both values according to the Eq.(6):

$$F(i,j) = 2. \frac{rec(i,j) * rec(i,j)}{rec(i,j) + rec(i,j)}$$

Based on this formula, the F-Measure for overall quality of cluster set C is defined by the Eq.(7):

$$F=\Sigma(C^i/N) . \max (F(i,j))$$

(b) Entropy: The entropy tells us how homogeneous a cluster is. The higher the homogeneity of a cluster, the lower the entropy is, and vice versa. Below is the Eq.(8)

$$\text{Entropy} = -\frac{1}{\log k} \sum C_j/N \sum (p_{ij} * \log p_{ij})$$

where $p_{ij}$ is the probability that a member of cluster $C_j$ belongs to class $C_i$.

(c) Purity: The cluster purity indicates the percentage of the dominant class members in the given cluster. For measuring the overall clustering Purity, we use the weighted average purity as shown below in Eq.(9):

$$\text{Purity} = \frac{1}{C_j/N} \max (\text{prec}(i,j))$$

Where $C_j$ is the number of clusters, and N is the total number of documents. Output parameters of the proposed algorithm in the table2 and table 3.

### TABLE II: OUTPUT PARAMETERS OF MESSAGE PASSING ALGORITHM

<table>
<thead>
<tr>
<th>dataset</th>
<th>Number of clusters</th>
<th>netsim</th>
<th>expref</th>
<th>Dpsim</th>
<th>Number of iterations</th>
<th>Elapsed time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohsumed</td>
<td>7</td>
<td>9.5952</td>
<td>.9523</td>
<td>8.6429</td>
<td>425</td>
<td>15.641sec</td>
</tr>
</tbody>
</table>

### TABLE III: OUTPUT PERFORMANCE PARAMETERS OF MESSAGE PASSING ALGORITHM.

<table>
<thead>
<tr>
<th>Number of documents</th>
<th>Entropy</th>
<th>F-measure</th>
<th>purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>30(three known classes)</td>
<td>0.22702</td>
<td>0.56654</td>
<td>0.76667</td>
</tr>
</tbody>
</table>

V. CONCLUSION & FUTURE WORK

In the proposed algorithm I tried to improve the solution quality of clusters which are produced by method called Message Passing. In the proposed algorithm similarities of data points for the same cluster are increased. By the experimental result, it is observed that the proposed algorithm gives 56.7% accuracy for datasets. From the net Similarity graph, it is concluded that the proposed algorithm increased net similarity. Some possible future work may includes: Optimize the computation process of similarities and further improve the efficiency; and find more practical method for setting preference values to speed up convergence of AP with no influence on performance. I am also interested to use this algorithm in applications where n-wise similarities are useful and for the sparse data where many data points can not be represented by many others as exemplars: i.e. $s(i,k) = -\infty$.

REFERENCES
