Efficient Similarity Measures for Clustering a Huge Dataset: A Critical Review

Desmond Bala Bisandu [a], Rajesh Prasad[b] and Musa Muhammad Liman[c]
[a,b,c]*School of IT & Computing, American University of Nigeria, Yola, Nigeria*
desmond.bisandu@aun.edu.ng, rajesh.prasad@aun.edu.ng, muhammad.liman@aun.edu.ng

Abstract. The need for appropriate applications of the various similarity measures for clustering has arisen over the years as data massively keeps on increasing. The issue of deciding which similarity measure is the best and on what kind of dataset have been a very cumbersome task in the field of data mining, data science, and organizations that are highly depending on the knowledge outcome from a huge set of data to make some vital / crucial decisions. Because various datasets portray some common features associated with them; therefore the need for clearer understanding of various similarity measures for clustering different datasets is needed. This paper presents a critical review of various similarity measures applied in text and data clustering. A theoretical comparison has been made to check the suitability of the measures on different kind of data sets.

Keywords**:** Similarity measures, document clustering, text document, Euclidean distance, and edit distance.

1. Introduction

Currently, we are faced with so much ever increased volume of documents that need to be operated upon before knowledge can be extracted from them [1]. There is abundant of documents moving over the Internet, huge document are collected over the digital libraries, repositories, and personal information that are digitized such as articles in the blog and emails which are quickly piling up every day [2]. These has pose a challenge on how efficient and effective we can organize the documents contents for easy access when need arise. In several applications, there is necessity to algorithmically measure the similarity between two strings comprising of symbols from alphabet in finite series [3]. For instance, identifying automatically the confusion between names of two drugs, it will be helpful to know that measure of similarity between *Tarodol* and *Tagredol* is higher than the similarity between *Tarodol* and *Indarel.* The problem of measuring the similarity between strings of a document occurs in different fields, which include speech recognition, bioinformatics, machine translation information retrieval, lexicography, and dialectology [3]. An issue related to computing the measure of similarity between texts as strings of words has been studied also.

Clustering has been a useful and important method that organizes automatically collection huge number of data objects into much smaller number of logical groups [3, 4]. For an example having an efficient document clustering, deciding appropriately the similarity measure to use has been proven to be a significant part of the clustering approach for some time now and also an interesting problem to research as well [2]. This is becoming more interesting even and demanding with World Wide Web development and the evolution of the Web 2.0. For instance, every results returned by the search engines are clustered to help focus the users search to relevant set of results. Many online stores customers have clusters such as the Amazon.com, to help provide recommendations collaboratively. In collaborative tagging and bookmarking, users of same clusters have same traits using their annotation for identification.

Document clustering in general groups documents with high similarity forming logical cluster, while documents with high dissimilarity are separated into different clusters. Though, the similarity definition of documents being similar or different has not always been clear and normally possess some variations with the actual setting of the problem [1]. For instance, to cluster research papers they are only regarded as similar if their thematic topics can be shared between them. Deploying clustering on the web sites, the in-

terested aspect is the clustering of the component pages according to the information type presented in the page. For example, dealing with university web sites, we may wish to separate between students and professors home pages, and research pages from courses pages. This clustering can cause more benefit analyzing further and dataset utilization such as retrieval of information and extraction of information, by similar types of information source grouped together.

Accuracy in clustering requires that closeness between pair of objects been defined precisely, in either pair wised or distance terms. So many similarity or distance measures have been proposed and applied widely, such as the Jaccard coefficient similarity and cosine similarity. In the meantime, similarity is mostly looked at in terms of dissimilarity or distance as well [5]. Pair wise distance calculation has been done using measure such as Euclidean distance or relative entropy and others [6]. The diversity of distance and similarity measures available for clustering documents, their effectiveness in any type of document clustering is still a challenge and not clear, although Akinwale et al. compared the effectiveness of some measures, but just focus on the n-gram measures basically and has not really discuss the basis of their existence [7].

In general, distance measure can be categorized into six (6) dimensions namely: Edit-based, Token-based, Hybrid, Structural (Domain-dependent), Phonetic, and Numeric [8, 9]. In this paper, we presented a systematic comparison of the various similarity measures on large text collection. The performances of the similarity measures based on the above six dimensions on various data sets are also analyzed. The strengths, weaknesses, space and time complexities of the above similarity measures are also presented. An experiment on R programming environment has been conducted on different dataset to decide on the applicability of different similarity measures.

Rest of the paper is organized as follows. Section 2 presents how important similarity measure is in clustering of text document and successful areas of application. Section 3 and Section 4 described the edit-based and the token-based similarity measures in details with solved problems for better demonstrations. Section 5 described other similarity measures in brief. Section 6 presents the results analysis, while Section 7 concludes the paper.

2. Similarity Measure and Its Importance

Generally text document files are unstructured which makes them not appropriate for regular databases. These documents contain various types of information, which understanding them automatically by direct clustering algorithms is very difficult [10]. Therefore, this unstructured terms in the documents should be converted to a format that is suitable to conduct some operations on them [4]. Mining useful information and pertinent data from variety of textual/ other types of document sources involves many kind activities, also extracting important knowledge from large data requires scalable analytics and intelligence services, techniques and good applications [2, 11] . However, little value has always been found in the extracted data in its row formats. Severally, web search have been confused with text mining. As much as they both result in data acquisition, the input has large gap. Users are dedicated towards specific data acquiring in web search, which mostly may entails specified and/ known data search [5, 8]. Many spell checking techniques have been discussed in the past without clearly stating their class based, some which are [9]: similarity keys, rule based techniques, n-gram based technique, neural network, and probabilistic techniques. The dimension reduction has been an significant preprocessing stage in the text grouping for analysis of the dimensions reduction for features space by removing the uninformative features [10].

Essentially some models for similarity measures are based on probability approach one of such is the n-gram, while other measures are not based on probability, but accurate measures [7]. These measures have been successfully apply in a wide variety of domains and problems such as compression of text, detection of spelling error and correction [12, 13]. Optical character recognition, retrieval of information [11, 12, 14], categorization of text automatically, representation of music, handwriting recognition and speech [15, 16]. Other domains that are useful include immunology computation, whole-genome protein sequence analysis [17]. Attribution of authorship, identification of language, tree reconstruction of phylogenetic, integration of data, cleaning and filtering, English language prediction [18], algorithms for phonetic matching, and retrieval text [19].

An example is the general string searching which has been done by some of the similarity measures such as n-grams [20], while others has been applied in some other are used for computer viruses detection

[21]. Many of these methods have been proven to be important in various tasks ranging from two texts comparisons to the level of quantification of the homology degree in the sequence. Problems in different research areas such computer science, biology, and operational research, etc. have applied some of the similarity measures without given specific concerns to the base of the similarity measures. Some token based measures have been used to grade text in mathematics [7]. Plagiarized text have been determined using some of these similarity measures (token-based) from a pool of METER corpus [19,20, 21, 22]. All research on the various similarity measures application have indicated that there is so much need for more research in making decisions on what type of similarity measure is best and the type of data set [7] .

Clustering a dataset is highly dependent on the similarity measures. The similarity measures indicate the level of nearness or severance between the data items, and this should be common to the characteristics of the clusters of the data item which distinguished them with all other clusters [23]. Severally these characteristics depend on either data or context of the problem present, there is no distance measure which have been separately agreed universally to be the best for all types of problems in clustering. Furthermore, to choose appropriately the similarity measure to use is a very paramount for the analysis of cluster, more especially for some unique kinds of clustering algorithms. Taking for example the clustering algorithms that are density-based, such as the DBScan [1], heavily depend on the computation of the similarity. Clusters are found to be as dense areas in data set using the density-based clustering, and the projected closeness of the matching data object to its adjacent objects is in turn the same as that of the given point. Noting that similarity/distance value is quantified by the closeness, it is clear that finding dense areas and projected clusters assignment of any new data object is dependent on the largeness of the similarity/distance computations. Therefore having a good understanding of the efficiency of the various measures is of enormous significance in choosing the best one.

Generally, distance/similarity measures indicates the similarity or the distance that exist between two distinct objects symbolic description into a numeric value, and this depends majorly on two factors- the two objects properties and their measures. In order to make the objective of this work clear we have conducted the review with some solve examples on some data set and the advantages and disadvantages of the similarity measures is also stated. These measures discussed below, different measures not only in distinct concluding representations, but also subject distinct requirements for the same clustering algorithm. Characteristics of distance measure are defined by the following metric:

Metric

Several distance measures exist but not all have been qualify to be a metric. There are generally four conditions that must be satisfied by any measure *d* to be qualified as a metric [1].

Let *d(a, b)* be the measure between any two objects *a* and *b* in a set.

1. The measure between any two separate points must be non negative, that is, $d(a, b) \geq 0$.
2. The measure between two separate objects must be zero if and only if the objects are identical, that is, $d(a, b) = 0$ for all $a = b$.
3. Distance measure must be symmetric, that is, distance from a to b should be same as distance from b to a, i.e. $d(a, b) = d(b, a)$.
4. The distance measure must satisfy the inequality of triangle, which is $d(a, c) \leq d(a, b) + d(b, c)$

3. Token-Based Similarity Measures

3.1. Cosine Similarity

Documents are mostly represented as a vector where each attribute is representing the frequency with which a particular term in a document occurs. The term vector representation of documents is used to find the similarity between the two corresponding documents and this show the correlation between the vectors. This is term as the cosine of the angles between two vectors, which is the so called Cosine similarity. This is one of the distance measure mostly apply in the clustering of text documents, such in retrieval of information applications [7] and clustering also [24].

Given two documents *a* and *b*, the cosine similarity between them can be defined using the following formula:

$$SIMcos\,(ai, bi) = \frac{\boldsymbol{ai} \,.\, \boldsymbol{bi}}{||\boldsymbol{ai}|| \times ||\boldsymbol{bi}||},$$

Where a_i and bi are vectors of m-dimensions over the document set *a* and *b*, i = 1, 2, 3, . . .,*N*. Each dimension of the document is a non negative. The co-

sine similarity is a non-negative value as a result and it's bounded between [0, 1].

The Cosine similarity has an important property which is its independence on the length of the document. For an example when you want to combine two documents of identical copies of a document *a* to have a new document *a'*, 1 will be the Cosine similarity between *a* and *a',* this mean that the two documents are identical. For now given other document *l, a, a'* the similarity measure will be same with the value *l*, which means that *sim (a, l)=sim(a', l)*. These mean that, documents that are having the same elements but differently in total are identically treated. This does not satisfy the metric second condition above; this is because after the whole combination the copies, the object will be different from the original document. Though, in practice, when normalizing two vector terms to a unit length such as 1, as in this case is the same for both *a* and *a'.* Example 1 given below illustrates this distance measure.

Example 1: Find the cosine similarity between the two documents a = *"Desmond"* and b = *"Diamonds"*
Step 1: Term Frequency (TF)
D, e, s, m, o, n, d = 2, 1, 1, 1, 1, 1
D, i, a, m, o, n, d, s = 2, 1, 1, 1, 1, 1, 1

Normalized Term Frequency of *a* and *b*
Using the formula NTF = $\frac{character/\,Term\ frequency}{Total\ number\ of\ characters/Terms}$
D = 2/7 = 0.285, e = 0.143, s = 0.143, m = 0.143, o = 0.143, n = 0.143 and
D = 2/8 = 0.25, i = 0125, a = 0125, m = 0125, o = 0125, n = 0125, s = 0125

Step 2: Inverse Document Frequency (TDF)
Using the formula IDF = 1 + $log_e \frac{Total\ number\ of\ documants}{number\ of\ documents\ with\ desired\ term\ in\ it}$
D = 1 + 1 + $\log_e \frac{2}{2}$ = 1 + 0 = 1, e = 1.6931, s = 1, m = 1, o = 1, n = 1, a =1.6931, i = 1.6931

Step 3: Term Frequency (TF) * Inverse Document Frequency (TDF) for *a* and *b*
D = 0.143 * 1, e = 0.242, s = 0.143, m = 0.143, o = 0.143, n = 0.143 and
D = 0.25, i = 0.211, a = 0.211, m = 0.125, o = 0.125, n = 0.125, s = 0.125

Step 4: Vector Space Model – Cosine similarity
$SIM_{Cos}\,(a, b) = \frac{dot\ product\ (a,b)}{||a||*||b||}$

Dot product (a, b) = (0.285 * 0.25) + (0.242 * 0) + (0.143 * 0.125) + (0.143 * 0.125) + (0.143 * 0.125) + (0.143 * 0.125) + (0.2116 * 0) + (0.2116 * 0) = *0.1428*
$||\ a\ ||$ = $((0.285)^2 + (0.242)^2 + (0.143)^2 + (0.143)^2 + (0.143)^2 + (0.143)^2)^{0.5} = (0.2215)^{0.5}$ = 0.4707
$||\ b\ ||$ = $((0.25)^2 + (0.2116)^2 + (0.2116)^2 + (0.125)^2 + (0.125)^2 + (0.125)^2)^{0.5} = (0.1519)^{0.5}$ = 0.3897
$||\ a\ ||\ *\ ||\ b\ ||$ = *0.4707 * 0.3897 = 0.1834*
$SIM_{Cos}\,(a, b) = \frac{0.1428}{0.1834}$ = *0.7781*

Advantages

It's very simple; the relevancy of tokens is better reflected. It's evaluation is very efficient. It is less sensitive towards swaps and gives value between [0, 1].

Disadvantages

Typographic errors between tokens are penalized, for example the similarity between 'sean bay' and 'shown boy' will be zero**.** The difference in ratings given to the items by various users is not taken into consideration in the computation.

3.2. Jaccard Coefficient Similarity

Jaccard coefficient, has been referred to as Tanimoto coefficient, is a similarity measure based on dividing the intersection by the objects union. Also it is applicable mostly in text clustering. Tanimoto coefficient uses a comparison between shared terms sum up the weight to the terms weight present in any of the document but not a shared terms between the two.

Given two documents a_i and b_i their Jaccard coefficient can be determine using the following formula:

$SIMjac(ai,\ bi) = \frac{|tok(a)\ \cap\ tok(b)|}{|tok(a)|+|\,tok(b)|-|tok(a)\ \cap\ tok(b)|} = \frac{|tok(a)\ \cap\ tok(b)|}{|tok(a)\ \cup\ tok(b)|}$

The measure of the similarity of the Jaccard coefficient ranges between 0 and 1. It becomes 1 if $a_i = b_i$ and 0 when both a_i and b_i are not having anything in common (disjoint), where 1 and 0 means the two objects are similar and completely dissimilar respectively. The distance measure corresponding to them is $D_J = 1 - SIM_J$ and D_J can be use in experiments.

Example 2: Given the two words terms Jones and Johnson, their tokens are *("Desmond, Diamonds"*) $|tok(a)\ \cap\ tok(b)|$ = 4,

$|tok(a) \cup tok(b)| = 13$
$SIM_{jacc}[a_i, b_i] = 4/13 = 0.3077$

Advantages

It's less sensitive to the word swaps, because it consider only whether token exist, not its position. Its evaluation is very efficient and is very simple to evaluate.

Disadvantages

Typographic errors between tokens are penalized, for example the similarity between '*sean bay'* and *'shown boy'* will be zero**.** Significance of the similarity measure is penalized in case of any error.

3.3. General n-gram Similarity

This algorithm was introduced by Niewiadomski. It measures to template string from an answer string as represented in the following formula [7]:

$SIM\ (a_1,a_2) = f(m_1,m_2) \sum_{i=m1}^{m2} \sum_{j=1}^{M-m+1} g(i,j)$

Where $f(m_1, m_2) = \frac{2}{(M-m1+1)(M-m2+2)-(M-m2+1)(M-m1)}$ represent the number of substring not shorter then m_1 and also not longer than m_2 in a_1, $g(i, j) = 1$ if and only if i – element-long substring of the string a_1 start from j-th position in a_1 at least appears once in the a_2 else $g(i, j) = 0$. If there exist a substring in other from one argument of comparison, the absolute similarity level will be measured as 1 which is identical of a_1 and a_2 [7]. $M(a_1)$, $M(s_2)$ = the length of string a_1 and a_2, M = max ($M(a_1)$, $M(s_2)$)

3.4. Bigram Similarity

The general n-gram measure has been used to derive several other gram algorithms including the bigram where n is equal to 2, therefore the formula is as follows: $SIM(a_1,a_2) = \frac{1}{(M-m+1)} \sum_{i=0}^{M-m+1} g(i) = \frac{1}{(M-2+1)} \sum_{i=0}^{M-2+1} g(i) = \frac{1}{(M-1)} \sum_{i=0}^{M-1} g(i)$

3.5. Trigram Similarity

The trigram is also a special type of n-gram where n is equal to 3, two strings a_1 and a_2 similarity can be determined by the following formula:

$SIM(a_1, a_2) = \frac{1}{(M-m+1)} \sum_{i=0}^{M-m+1} g(i) = \frac{1}{(M-3+1)} \sum_{i=0}^{M-3+1} g(i) = \frac{1}{(M-2)} \sum_{i=0}^{M-2} g(i)$

3.6. Oddgram Similarity

This is a special type of n-gram that takes m(m-1)/2 processing substrings before the performance is measure. This take the n-gram matching substrings by half for processing which still reduce running time. The match strings for the method is denoted by a_1, a_2 and $max(M(a_1), M(a_2)) = M$ which represent the maximum between the two strings a_1 and a_2. If M is an odd number then $M = \lceil \frac{M}{2} \rceil$

$SIM\ (a_1, a_2) = \frac{1}{M^2} \sum_{i=M}^{M} \sum_{j=1}^{M-i+1} g(i,j)$ otherwise $\frac{1}{(M^2+M)} \sum_{i=M}^{M} \sum_{j=1}^{M-i+1} g(i,j)$

3.7. Sumsquare Gram Similarity

This has it root from the n-gram measure in which the time for processing is quadratic for all the n-gram in the line statement query. Even though n-gram similarity measure are easy to manage and generate, their time and space complexity are quadratic and therefore good for both sumsquare gram and oddgram since they work in quadratic. The sumsquare gram and oddgram measure write their results in pair submissions (text and pattern matching). Given pattern and text matching *i* and *j*, a_{ij} is close to 1 only if they are both identical and also close to 0 if they are very different. This means that both oddgram and sumsquare grams normalized form falls in the interval [0, 1]. Likewise oddgram and sumsquare grams measure of similarity are symmetrically expected, meaning $a_{ij} = a_{ji}$ will hold for all *i, j*. The matched string for sumsquare denoted by a1, a2 and $max(M(a_1), M(a_2)) = M$ taking the maximum length of the two string a_1 and a_2. $M = \lfloor \sqrt{M} \rfloor$

$N = times\ to\ jump = M - 1$

$Q = first\ jump = M^2 - (M-1)^2$

$SIM_{sq}(a_1, a_2) = \frac{6}{M(M+1)(2M+1)} \sum_{i=1}^{Q} \sum_{j=1}^{M} g(i,j)$

3.8. Set-based Trigram Similarity

This similarity measure technique is derived from the set theory of similarity measure. This is concern with measuring similarity in terms of number of common trigram of two set of entities. The string sharing of both pattern and text matching weight are increased by three times. This is asymmetric since it does not give consideration to (false, false) to be a matching pattern. It is represented by the following formula:

S_{BT} *: T(A, B)* = $\frac{3(trigram(A \frown B))}{trigram(A)+ trigram(B)+trigram(A \frown B)}$

Examples 3: given a_1 = *Desmond*, a_2 = *Diamonds*. $M(a_1)$ = 7 and $M(a_2)$ = 8, Max {$M(a_1)$; $M(a_2)$} = 8

a_2 occurs in the substring of a_1 as follows:

1-element D, m, o, n, d= 5
2-element Dm, mo, on, nd = 4
3-element Dmo, mon, ond = 3
4-element Dmon, mond = 2
5 -element Dmond = 1

1. General n-gram measure
   SIM(a_1, a_2) = $\frac{2}{M^2+M}$ $\sum_{i=m1}^{m2}\sum_{j=1}^{M-m+1} g(i,j)$ = $\frac{2}{8^2+8} \times \frac{5+4+3+2+1}{1} = \frac{2*15}{72} = 0.4166$
2. Bigram measure
   SIM (a_1, a_2)= $\frac{1}{(M-m+1)}\sum_{i=0}^{M-1} g(i) = \frac{1}{(8-1)} \times \frac{4}{1}$ *= 0.7777*
3. *Trigram measure*
   SIM (a_1, a_2) = $\frac{1}{(M-m+1)}\sum_{i=0}^{M-2} g(i) = \frac{1}{8-2} \times 3$ $= \frac{3}{6}$ *= 0.5000*
4. *Oddgram* measure *If M is odd then M=* $\lceil \frac{M}{2} \rceil$ *= M = even =* $\lceil \frac{8}{2} \rceil$ *= 4,*
   SIM (a_1, a_2) = $\frac{1}{(M^2+M)}\sum_{i=M}^{M}\sum_{j=1}^{M-i+1} g(i,j)$ $= \frac{1}{(4^2+4)} \times \frac{4+2+1}{1} = \frac{7}{20}$ *= 0.3500*
5. Sumsquare gram measure = *M* = $\lfloor\sqrt{8}\rfloor$ = 2 = *time to jump* = M – 1=2 – 1= 1 , *Q = first jump* = $M^2 - (M-1)^2 = 2^2 - 1^2$ =3, $1^2 - 1^2$= 0
   SIM_{sq} (a_1, a_2) = $\frac{6}{M(M+1)(2M+1)}\sum_{i=1}^{Q}\sum_{j=1}^{M} g(i,j)$ = $\frac{6}{2(3)(5)} \times \frac{5+0}{1}$ $= \frac{5}{5}$ *= 1.0000*
6. Set-based trigram measure S_{BT} *: T(*a_1*,* a_2*)* = $\frac{3(trigram(A \frown B))}{trigram(A)+ trigram(B)+trigram(A \frown B)}$ = $\frac{3*3}{6+6+3}$ $= \frac{9}{15}$ *= 0.6000*

Advantages

This uses overlapping which reduces the effect of the typographical error in the computation**.** It's very efficient and always captured automatically the most frequent root string.

Disadvantages

Some n-grams such as sumsquare gives approximated result**.** It is very difficult to compute and evaluate. It is time consuming.

4. Edit-Based Similarity Measures

4.1. Levenshtein Similarity

This is the distance between two strings which is based on the minimum number of operations; the operations are insertions, deletions, and substitutions which are required in transforming a string into the other second string, this is also referred to as the Levenshtein distance and it exist in two variants which are: Optimal string alignment and Demarau-Levenshtein. Both the algorithms can do the same thing as Levenshtein except they can accomplish transposition too. The difference between Optimal String Alignment and Damerau-Levenshtein is that Optimal String Alignment Algorithm only completes transposition under the condition that no substring is edited more than once whereas Damerau-Levenshtein is not restricted by such a thing. That is also why Optimal String Alignment is sometimes called the Restricted Edit Distance. This aim at finding the edit distance between two strings, together with their optimal transcript which describes the transformation that take place [25]. **Insertion operation:** Replace the blank by the desired character i.e a δ (∈, a); ∈ ~ a, **Deletion operation:** Replace the desired character by blank i.e ∈ δ (a, ∈); a ~ ∈, **Replacement operation:** Replace the desired character by the other character needed i.e a by b; δ (a, b); a ~ b. This algorithm com-

pute transcript base on the dynamic programming algorithm, the principle of optimality of this algorithm is that; the best overall solution must always contain the best transcript of the two substrings. The algorithm can be represented as follows:

- Initialize matrix *M* of size $(|s_1| + 1) \times (|s_2| + 1)$
- Fill matrix: $M_{i,0} = i$ *and* $M_{0,j} = j$
- Recursion: $M_{i,j} = \{ Mi\text{-}1,j\text{-}1 \text{ if } x[i] = y[i] \text{ else } 1 + \min(Mi - 1, j, Mi, j - 1, Mi - 1, j - 1)$
- Distance: LevenshteinDist $(x, y) = M_{|x|,|y|}$

Levenshtein Similarity: $SIM_{Levenshtein}(a_1, a_2) = 1 - \frac{LevenshteinDist(a1,a2)}{max(|x|,|y|)}$

Example 4: Given a_1 = ***Desmond*** *and* a_2 = ***Diamonds***, the Levenshtein distance = 3 the Levenshtein similarity measure is given as:

$SIM_{Levenshtein}(a_1, a_2) = 1 - \frac{3}{max(|7|,|8|)} = 0.6250$

Advantages

It is simple and fast. It is highly applicable on small and big strings. It is not restricted to strings having same length and the cost at most can be the length of the longest string.

Disadvantages

Running Levenshtein on two long strings results in a long time and a big cost that is proportional to the product of the two string lengths

4.2. Needleman-Wunsh Similarity

This is also called the optimal alignment and was introduce in 1970, the algorithm model variations between more explicitly strings than a series of transformation. This algorithm is an application of best path strategy (Dynamic programming) used in finding optimal alignment of sequence.

The basic idea behind this algorithm is from the fact the any path that lead to an optimal path point is also an optimal path. Hence the optimal path is a collection of several sub-path, in this algorithm the optimal path take from the beginning to the end of both the sequence involved, which bring about the concept of 'global alignment'[26]. The similarity score alignment is computed as for example given that the alignment between strings *x and y* is *A*, a score matrix $C(x_i, y_j)$ and gap penalty *cg*, the score of the *(x, y)* given the *A* is the sum of the score of all matches in the A minus the penalties for the gap. Dynamic Programming can be applied to a large search space that can be structured into a succession of stages such that:

- The initial stage contains trivial solutions to sub-problems
- Each partial solution in a later stage can be calculated by recurring on only a fixed number of partial solutions in an earlier stage.
- The final stage contains the overall solution.

This algorithm has three basic stages in the computation of it similarity measure as stated below:

- Initialization
- Matrix fill or scoring
- Trace back and alignment

The computation of the Needleman-Wunsch score is done by the following formula:

$S(i, j) = max(s(i\text{-}1, j\text{-}1) + c(x_i, y_j), s(i\text{-}1, j) + cg - (gap) - 1, s(i, j\text{-}1) + cg))$

Initially $s(0, j) = -jcg$

$s(i, 0) = -icg$

Example 5: One possible alignment between a= ***Desmond*** and b = ***Diamonds***, showing (D e s m o n d - = D i a m o n d s). Assign score for a correspondence between every pair of characters; penalizes transformations on a case by case basis. A correspondence of two identical characters may score 2; or -1 otherwise and a gap penalty A gap of length 1 has a penalty cg (ex: 1); a gap of length k has a penalty k*cg, therefore 2 (for match D-D) – 1 (for match e-a) - 1 (for match s- a) + 2 (for match m-m) + 2 (for match o-o) + 2 (for match n-n) + 2 (for match d-d) - 1 (for gap s) = 1, 2 – 1 – 1 + 2 + 2 + 2 + 2 – 1 = 4 – 3 = 7,

$SIM_{Nw}(a, b) = 1 - \frac{1}{7} = 0.8571$

Advantages

It is a fast algorithm for sequence alignment. It is efficient. It allows different penalties for variations between strings. It gives the global score of the string and therefore increases the general scope. It explicitly models gaps in the alignment of the two strings, and assigns an arbitrary cost to the gap.

Disadvantages

It is slow to compute. It consumes a lot of space and is difficult to evaluate

4.3. Jaro-Winkler Similarity

This similarity measure works on the number of mismatch between two strings by character transpositions been allowed a type of edit operation that performs comparison using dynamic programming principles [9]. The Jaro was invented in the year 1989 and it is essentially used to compare two strings a_1 and a_2 by common character identification among the two strings. The characters are said to be common to the two strings if and only if they appear in equal positions within the two strings, this is respectively denoted by *i and j*, not different from the h of the shorter string. This is formally represented as $|i - j| \leq 0.5 \times \min(|a_1|, |a_2|)$. Immediately all characters common are identified, both a_1 and a_2 are sequentially traversed, and determining the number *t* of transpositions of the common characters where when *i -th* common character of a_1 is not equal to the *i –th* common character of a_2 then transposition occurs. Give the number of transposition *t* set δ of common characters, the Jaro similarity measure can be computed using the following formula:

$$SIM_{Jaro}(a_1, a_2) = \frac{1}{3} \times \left(\frac{|\delta|}{a1} + \frac{|\delta|}{a2} + \frac{|\delta| - 0.5t}{|\delta|}\right)$$

Example 6: Given a_1 = ***Desmond*** and a_2 = ***Diamonds***. Then $|a_1| = 7$ and $|a_2| = 8$. The maximum distance between characters that are common is $0.5 \times \min(7, 8) = 3.5$. The common character set is $\delta = \{D, m, o, n, d\}$, where _ is the space character. None of the matching line crosses another matching lines, indicating that none of the common characters produces a transposition, making have *t = 0.* Then the Jaro distance is as thus:

$$SIM_{Jaro}(a_1, a_2) = \frac{1}{3} \times \left(\frac{5}{7} + \frac{5}{8} + \frac{5-0}{5}\right) = 0.7798$$

This algorithm is generally performing well on strings that have slight spelling variations. But because of the restriction on common characters have to occur in certain distance from each other. , this does not cope well with longer strings separating common characters. This problem mostly occurs for names with common prefix but one name has an additional suffix (e.g Desmond B vs. Desmond Bala stored as first name). This is what because the extension of this algorithm to a new modified one called Jaro-Winkler similarity measure invented in 1991, given consideration of two special strings a_1 and a_2 with a common prefixδ, the Jaro-Winkler similarity can be computed with following formula:

$SIM_{Jaro\text{-}Winkler}[a_1, a_2] = JaroSim(a_1, a_2) + |\delta| \times f \times (1 \text{ JaroSim}(a_1, a_2))$, where *f* is a constant scaling factor of how much corrected upward based on the common prefix δ of the similarity measure.

Example 7: Given a_1 = ***Desmond*** and a_2 = ***Diamonds***, there are 5 common characters between the two strings corresponding to the common prefix δ = *peter.* There are no permutation of characters as clearly indicated in δ so *t = 0.* Then the $SIM_{Jaro}(a_1, a_2) = 0.7798$. By the assumption that the scaling factor *f = 0.1*the Taro-Winkler similarity measure is: $SIM_{Jaro\text{-}Winkler}(a_1, a_2) = 0.7798 + 5 \times 0.1 \times (1 - 0.7798) = 0.8899$

Advantages

It is very accurate. It is very efficient and is very flexible to accommodate variation in strings

Disadvantages

This does not cope well with longer strings separating common characters. It reduces the score of similarity in case of a longer string

5. Other Similarity Measures

Some measures are mostly used for duplication detection, buy none is applicable conceivable to all scenarios. As seen some are special for short strings and others are for long strings with few typographical errors while others are insensitive to word swaps [9]. There is need to introduce some other special cases similarity measures i.e phonetic, numeric, structural, hybrid similarities respectively.

5.1. Phonetic Similarity

This is a similarity measure that focuses spoken words sounds, which might be very similar despite the large differences in the spelling. For example, the two strings *Czech* and *Cheque* are not very similar, but however they are hardly distinguishable phonetically. Hence they are said to have a large phonetic similarity.

A very common phonetic coding scheme is known as Soundex, and the computational idea of the phonetic similarity is first transforming strings(or tokens)

into their various phonetic representation and the similarity measure is then applied on the strings or tokens on Soundex representation [9].

5.2. Numeric Similarity

This is the similarity measure applied to numeric data. Normally, numbers are considered as strings, that yields unsatisfactory results, for example when comparing *1999* and *2000*. The measure of the difference between the two numbers compared is a solution, i.e *[1999-2000]*. Though in different domains, there are different meanings to difference in numbers. For example, a microscopic scale difference measurement, a 1mm is a very large difference, but on macroscopic it is almost nothing. A "normalize" way possible is taking the distribution of values in the domain into account [9].

5.3. Structural Similarity

This similarity measure takes the structure of the data into consideration, with no just focus on the content as seen in the other similarity measures. Though, putting the structure into consideration may be of great relevant, e.g when comparing XML data trees. The widely used similarity measure is the *trees edit distance* and the variation thereof [9]. Importantly, the tree edit distance is also an edit-based measure of similarity (such as levenshtein distance), but does not edit operations on characters except on all the strings or token structure.

5.4. Hybrid Similarity

This similarity measure is combining both the tokenization and string similarity score. This algorithm is referred to as hybrid because of the feature of both tokenization and string similarity combination. The extended Jaccard similarity, and Monge-Elkan similarity are types of this similarity measure, because they includes similar tokens in set of overlapping descriptive data [9].

6. Result and Conclusion

6.1. Result

Table 1 below show the comparison of the similarity measures based on some properties. As it is indicated in the table below Needleman-Wunsh and the Jaro-Winkler measures are having worst complexities in both time and space with high space consumption and low overall performance. The n-gram has the second highest complexity both in time and space, but having moderate use memory in execution. On average the Jaccard and Cosine have similar complexity in time and space with low consumption of memory and high performances rate.

6.2. Conclusion

This paper presents a rigorous theoretical review on the various similarity measures used in clustering the huge dataset. Similarity measure provides the level to which documents are near or far from each other. From the study conducted, it is shown that the most of edit-based similarity measures have higher complexities in both time and space than token-based**,** as a result of these complexities the memory consumption of the edit-based measures are higher than the token-based. From the study conducted its clearly shown that Levenshtein, Needleman, and Jaro measures are applied in mainly text documents, while the cosine, Jaccard, are applied mainly in documents collection, and the n-grams measures are for both text and document datasets. Also it is recommended from the study the token-based similarity measures are applied in clustering huge document of different kind, and the edit-based are applied to text documents in specific.

References

1. A. Huang, Similarity measures for text document clustering, In proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand, pp. 49–56, Apr., 2008.
2. V. Medvedev, O. Kurasova, J. Bernatavičienė, P. Treigys, V. Marcinkevičius, and G. Dzemyda, A new web-based solution for modelling data mining processes. Simul. Model. Pract. Theory., vol. 12, no. 1, pp. 53-60, Mar., 2017.
3. G. Kondrak, N-gram similarity and distance. In International Symposium On String Processing and Information Retrieval, pp. 115–126, 2005.
4. S. N. Bharath Bhushan and A. Danti, Classification of text documents based on score level fusion approach. Pattern Recognit. Lett., vol. 94, no. 3 pp. 118-126, Jul., 2017.
5. H. Hashimi, A. Hafez, and H. Mathkour, Selection criteria for text mining approaches. Comput. Hum. Behav., vol. 51, pp. 729–733, Mar., 2015.
6. S. A. Shehab, A. Keshk, and H. Mahgoub, Fast dynamic algorithm for sequence alignment based on bioinformat-

ics. Proc. Int. J. Comput. Appl., vol. 8, no. 7, pp. 0975-8887, 2012.
7. A. Akinwale and A. Niewiadomski, Efficient similarity measures for texts matching, J. Appl. Comput. Sci., vol. 23, no. 1, pp. 7–28, 2015.
8. F. Naumann, Similarity measures, Inf. Syst., vol. 38, no. 6, pp. 887-907, Sep., 2013.
9. F. Naumann and M. Herschel, An introduction to duplicate detection. Morgan & Claypool Publishers, 2010.
10. L. M. Abualigah, A. T. Khader, M. A. Al-Betar, and O. A. Alomari, Text feature selection with a robust weight scheme and dynamic dimension reduction to text document clustering, Expert Syst. Appl., vol. 84, pp. 24–36, Oct., 2017.
11. D.W. Kim, Book review, Comput. Sci. Rev., vol. 5, no. 2, pp. 163–203, May, 2011.
12. S. Kaur and Prof. Kiranjyoti, Mining text using levenshtein distance in hierarchical cluetering.pdf, Int. J. Comput. Tech., vol. 2, no. 1, pp. 15, 2015.
13. A. Akinwale and A. Niewiadomski, "Effective similarity measures in electronic testing at programming languages," Journal of Applied Computer Science, vol. 20, no. 2, pp. 7-20, 2012.
14. E. Ménard and M. Smithglass, "Digital image description: a review of best practices in cultural institutions," Libr. Hi Tech., vol. 30, no. 2, pp. 291–309, 2012.
15. V. Hollink, J. Kamps, C. Monz, and M. De Rijke, Monolingual document retrieval for European languages, Inf. Retr., vol. 7, no. 1, pp. 33–52, 2004.
16. W. B. Cavnar, J. M. Trenkle, and others, N-gram-based text categorization, Ann Arbor MI, vol. 48113, no. 2, pp. 161–175, 1994.
17. S. Mohamed, D. Rubin, and T. Marwala, Multi-class protein sequence classification using fuzzy ARTMAP, Systems, Man and Cybernetics, SMC '06. IEEE International Conference on, vol. 2, pp. 1676–1681, Oct., 2006.
18. J. Xu et al., Self-Taught convolutional neural networks for short text clustering, Neural Netw., vol. 88, pp. 22–31, Apr., 2017.
19. G. E. Kopec, M. R. Said, and K. Popat, N-gram language models for document image decoding, in Proceedings of the IS&T/SPIE 14th Annual Electronic Imaging Symposium, Vol. 4670, pp. 191–202, Jan., 2002.
20. C. Mayers and D. L. Whiting, Data compression apparatus and method using matching string searching and Huffman encoding. Google Patents, Jul., 1996.
21. T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, Detection of New Malicious Code Using N-grams Signatures, in PST, vol. 2, pp. 41-42, Sep., 2004.
22. A. Barrón-Cedeño and P. Rosso, On automatic plagiarism detection based on n-grams comparison, in European Journal of Information Retrieval Research, vol. 6, no. 4, pp. 51–73, Apr., 2009.
23. A. Barrón-Cedeño, M. Potthas, P. Rosso, B. Stein, and A. Eiselt, Corpus and evaluation measures for automatic plagiarism de tection, in Department of Information Systems and Computation, research, lrec, 2010.
24. Y. Cao, P. Zhang, J. Guo, and L. Guo, Mining large-scale event knowledge from web text, Procedia Comput. Sci., vol. 29, pp. 478–487, 2014.
25. S. Dana, S. James, Large Edit distance with multiple block operations, International Symposium on String Processing and Information Retrieval, vol. 28, no 57, pp. 369-377, 2003.
26. C. Kacfah Emani, N. Cullot, and C. Nicolle, "Understandable Big Data: A survey," Comput. Sci. Rev., vol. 17, pp. 70–81, Aug., 2015.
27. C. Peter, A comparison of personal name matching: techniques and practical issues, The Australian National University, Joint Computer Science Technical Report Series Department of Computer Science Faculty of Engineering and Information Technology Computer Sciences Laboratory Research School of Information Sciences and Engineering, Sep.2006.

Table 1: Comparison of the various similarity measures

S/ no	Measures/ Properties	Time Complexity	Space Complexity	Category of Measure Based	Memory Space Consumption	General Performance Rating	Reference	Type of data
1	Cosine	O(n)	O(n)	Token-based	Low	High	[7,24]	Documents
2	Jaccard	On(1)	On(1)	Token-based	Low	High	[22]	Documents
3	n-gram	O(mn)	$O(m^3)$	Token-based	Moderate	Moderate	[7]	Documents and text
4	Levenshtein	O(|m| . |n|)	O(min(|m|, |n|))	Edit-based	Low	High	[25]	Text
5	Needleman-Wunsh	O(mn) quadratic	O(mn) quadratic	Edit-based	High	Low	[6,26]	Text
6	Jaro-Winkler	O(|m| + |n|) quadratic	O(|m| + |n|) quadratic	Edit-based	High	Low	[6,9]	Text

m= Number of terms / characters, n= Size of the data set/ document