Document clustering with evolved multi-word search queries where the number of classes is unknown.

Laurence Hirsch[1] : Sheffield Hallam University, Sheffield, UK, S1 1WB

Robin Hirsch: University College London

Bayode Ogunleye: University of Brighton

[1] Corresponding author email: l.hirsch@shu.ac.uk

Document clustering with evolved multi-word search queries where the number of classes is unknown.

Laurence Hirsch[1] : Sheffield Hallam University

Robin Hirsch: University College London

Bayode Ogunleye: University of Brighton

Abstract

We present a novel, hybrid approach for clustering text databases. We use a genetic algorithm to generate and evolve a set of search queries in Apache Lucene format. Clusters are formed as the set of documents matched by a search query. The queries are optimized to maximize the number of documents returned and to minimize the overlap between clusters (documents returned by more than one query). Where queries contain more than one word, we have found it useful to assign one word to be the root and constrain the query construction such that the set of documents returned by any additional query words intersect with the set returned by the root word. Multiword queries

[1] Corresponding author.

are interpreted disjunctively. We also describe how a gene can be used to determine the number of clusters (*k*).

Not all documents in a collection are returned by any of the search queries in a set, so once the search query evolution is completed a second stage is performed whereby a KNN algorithm is applied to assign all unassigned documents to their nearest cluster. We describe the method and present results using 8 text datasets comparing effectiveness with well-known existing algorithms. We note that search query format has the qualitative benefits of being interpretable and providing an explanation of cluster construction.

Keywords

Document clustering, search query, genetic algorithm, machine learning, Apache Lucene.

1 Introduction

Clustering algorithms group a collection of documents into subsets or clusters to enable users to explore, organise, summarise, curate and visualise large volumes of text. Documents within a cluster should be similar to each other (cohesion) whilst documents in different clusters should be dissimilar (separation). Text clustering is a crucial component of text mining as it involves classifying and grouping similar content.

For automated clustering, documents are traditionally represented by a multi-dimensional feature vector where each dimension corresponds to a weighted value of a term within the document collection [1]. Various similarity or distance measures have been proposed and are a central component of most text clustering algorithms. Using such a method it is often difficult for a human to understand how the clustering is performed and there has been some criticism of the black box nature of many successful machine learning models, particularly where large datasets may contain human biases and prejudices [2, 3]. As a result of the risks associated with relying on sophisticated machine learning text clustering models which we do not really understand, significant have efforts have been made to create explainable systems. Work has been done on alternative models which recognise word order such as using lexical chains to preserve the semantic relationships between words, for example by using WordNet [4, 5]. Efforts have also been made to generate a set of human interpretable rules from 'black box' systems such as support vector machines as in [6]. Another effective approach has involved combining neural network models with symbolic representations [7].

Most text clustering systems require that the user provide the number of required clusters (*k*) in advance. However, *k* is often not known. We also present a solution to the harder problem of discovering *k* as well as defining clusters.

1.1 Motivation

When constructing a search query, a human user will normally try to find a word or combination of words that returns the documents they are interested in but will not return other documents. In their seminal work Manning et al. assert that the '*fundamental assumption'* of information retrieval is what they term the 'cluster hypothesis':

> *Documents in the same cluster behave similarly with respect to relevance to information needs. The hypothesis states that* ***if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant[2].*** *"* [8]

Following this hypothesis, we have developed a system called eSQ (evolved Search Queries), which uses a Genetic Algorithm (GA) for evolving a set of search queries where a cluster is the documents returned by a single query in the set. The overall objective is to develop an effective clustering

[2] Our emphasis

system with a natural fit for information retrieval needs and with the following desirable characteristics:

1. easily interpreted by a human.
2. modifiable by a human.
3. provide a causal explanation of cluster construction.

1.2 Contribution

We believe this is the first attempt to produce effective document clustering based on simple, explainable multi-word search queries. Furthermore, we also believe that this is the first attempt to use a GA to determine the number of clusters required. The techniques presented should be useful in both text mining and the development of new methods of clustering.

2 Background

Many algorithms have been proposed to achieve document clustering but the most popular is the k-means algorithm. K-means begins with *k* arbitrary centres, typically chosen uniformly at random from the data points. Each point is then assigned to the nearest centre, and each centre is recomputed as the centre of mass of all points assigned to it. These two steps are repeated until the process stabilizes. K-means++ is an enhanced version of the k-means algorithm and uses a randomised seeding technique which is a specific way of selecting initial centres [9, 10]. K-means (and its variants) is still widely used and an active topic of current research [11]. The main disadvantage of k-means is that we must provide the desired number of clusters to the algorithm in advance. Furthermore, clustering algorithms such as k-means are limited in their ability to capture contextual information and semantically explain the reasoning behind the clustering results.

Agglomerative clustering is a popular alternative to k-means which can determine a suitable value for *k*. Agglomerative clustering is the bottom-up form of hierarchical clustering which treats each data point or object as cluster at the initial stage [12]. In subsequent iterations, using a linkage function the cluster joins the nearest cluster at distance D. The distance D is calculated using metrics such as Euclidean distance. The iteration continues until a large cluster with all the objects is formed [12]. There have been attempts to combine agglomerative clustering and k-means [13].

The system we propose uses a GA which is a stochastic global optimisation technique which mimics the process of Darwinian evolution whereby the search for solutions is guided by the principles of selection and heredity [14]. GAs have proved to be an effective computational method, especially in situations where the search space is un-characterized (mathematically), not fully understood, or/and highly dimensional [16]. Clustering a large volume of text is an example of unsupervised problems that fits all these characteristics and GAs have been used here, often as a means of optimizing the allocation of cluster centres [17, 18, 19]. Frequently, each chromosome represents a combination of centres which represent the candidate solution to the clustering problem [11]. Commonly, a ratio of the intra-cluster distance of clusters against the inter-cluster distances between the cluster can be used. Document clustering using a fitness function based on the concept of nearest neighbour separation has also been proposed [20].

GAs have also been in use for some time to generate rules for text classification [21, 22, 23] and clustering [11, 24, 25], which have the advantage of being explainable. The simple disjunctive search queries produced by the eSQ system are easy to understand and are potentially modifiable by a human analyst. The eSQ system presented here also has a novel fitness test based entirely on the count of unique query hits. We also describe how the query-based clustering system has been combined with a second stage where a KNN classifier is used to assign documents not belonging to any query generated cluster to their nearest cluster.

3 Materials and Methods

3.1 Apache Lucene

Apache Lucene is a high-performance full-text indexing and searching software library written in Java. Evolutionary computation is notoriously resource-intensive, and we find that using Lucene to store and query the document collections is a significant boost to the performance of the system.

3.2 Document Collections

Different clustering algorithms can produce divergent results when compared to each other on different datasets with different types of text. We, therefore, ran our experiments on 8 different datasets selected from 3 document collections containing very different types of document. Each dataset is labelled in bold.

CrisisLex

An increasing number of short texts are being generated and it has been noted that this environment is complicated by sparsity and high-dimensionality, meaning that the vector space model and normal text clustering methods may not work well [26, 27]. CrisisLex.org is a repository of crisis-related social media data and tools [28]. The 'CrisisLexT6' collection2 contains tweets collected in 2012-13 in different crisis situations. We use 1000 of the tweets from each of the categories. **Crisis3** is created from: Colorado wildfires, Boston bombings and Queensland floods. **Crisis4** is created from Colorado wildfires, Boston bombings, Queensland floods and LA airport shootings.

20 Newsgroups

In the 20 Newsgroups collection [29] documents are messages posted to Usenet newsgroups, and the categories are the newsgroups themselves. The data on this set is considered particularly noisy and as might be expected does include complications such as duplicate entries and cross postings. We create three datasets from this collection by randomly selecting 400 documents from each of the categories. **NG3** is created from: rec.sport.hockey, sci.space and soc.religion.christian. **NG5** is from: comp.os.ms-windows.misc, misc.forsale, rec.sport.hockey, sci.space, soc.religion.christian. **NG6** is from: comp.graphics, rec.sport.hockey, sci.crypt, sci,space, soc.religion.christian, talk.politics.gun as in [18].

Reuters-21578

Reuters-21578 news collection contains news articles collected from the Reuters newswire in 1987. We create three datasets using 200 documents from each category. **R4** contains documents from crude, earn, grain, money-fx. **R5** contains documents from: coffee, crude, interest, sugar, trade. **R6** contains documents from acq, crude, earn, grain, money-fx and ship as used in [18]

3.3 Method

We use a GA to specify a set of search queries. The documents returned by each query is a cluster. We use a simplified example based on the problem of clustering documents in the Newsgroup 5 (NG5) dataset to assist the explanation. To begin we assume the simpler case of evolving single word queries. We will then go onto to explain the extra requirements needed when building multiword queries.

Step 1: pre-processing

Before we start evolving queries, all the text is placed in lower case and a small stop set is used to remove common words with little semantic weight. For each dataset, a Lucene index is constructed from the collection of documents, and each document labelled (using Lucene fields) according to its pre-set category. Of course, the GA has no access to the category label which is only used to evaluate the effectiveness of the clustering once all the stages have completed.

Step 2: create a wordlist

In the second step, we create an ordered list of significant words which is used by the GA for building queries. To construct the list, the TF*IDF (term frequency * inverse document frequency) value for each term in the collection is calculated. TF-IDF (often used in term weighting) is used to identify terms that are concentrated in particular documents and may therefore be of more significance in a collection. TF is the number of occurrences of a term in a document and IDF is the inverse of the number of documents in which the term occurs. For each term in the index, we determine TF*IDF values occurring in each document as indicated in the groovy style pseudo code below, where *terms* is the set of terms and *documents* is the set of documents in the index.

```
Map<Term, Double> tMap = [:]
terms.each {term ->
  double tfidfTotal = 0
  documents.each {doc ->
    tfidfTotal += tf * idf
  }
  tMap.put(term, tfidifTotal)
}
TermList tList= tMap.sort{it.value}.keySet
```

Terms are sorted by their overall TF*IDF value, and the top 100 words are selected for use in query building. This step is only required once for each index before the start of the evolution, after which the list is fixed. The integer index is simply the words place in the TF*IDF ordering. In the example shown in Table I the length of the list is only 8.

Table 1. word list

0	1	2	3	4	5	6	7
space	nasa	god	orbit	hockey	file	sale	game

Step 3: determine *k* (the number of categories).

If *k* is predefined, then this step can be omitted. If *k* is genome defined, we include an extra gene in the genome for this purpose. An int value in the inclusive range [2 .. 9] (8 possible cluster sizes) is used.

Step 4: Create generation 0

Table II shows a sample chromosome from the population of generation 0. Chromosomes have an integer representation where the values can be in the range [0 .. 100] (the maximum size of the wordlist). In the example below each gene defines a single word search query (SQ) and each search query defines the cluster as those documents which contain that word

Table 2. Creating single word search queries

	SQ0	SQ1	SQ2	SQ3	SQ4
Chromosome:	0	4	5	1	7
Query Words:	space	hockey	file	nasa	game

In the example above the number of clusters (*k=5*) is determined from the known number of categories before the start of the evolution. However, as mentioned above we can optionally add another gene in the chromosome to set a value for *k* (the number of clusters).

Step 5: fire each query in the set.

In our example, five single word search queries are generated for the NG5 dataset. For each individual in the population fire each of the search queries and determine its fitness by examining the clusters of documents returned by the queries and counting the number of documents returned which occur in only 1 cluster (see fitness calculation below).

Step 6: apply genetic operators

Apply genetic operators to create a new generation.

Step 7: repeat

Repeat steps 5-6 for 120 generations (termination criteria) and select the individual with the highest fitness.

Step 8: create training documents

The selected individual at the end of a run will produce a set of single word search queries. Fire each of these queries and save the document clusters produced as training documents. Remove any documents which are returned by more than 1 query.

Step 9: KNN

Some documents in the collection may not contain any of the query words or are returned by more than one query and are therefore not included in any of the clusters. We use the training documents from step 8 together with the Lucene implementation of the K-Nearest Neighbour (KNN) algorithm to add each unassigned document to its closest cluster.

Step 10: **evaluation**.

If evaluation is required measure the final V Measure and Adjuster Rand Index value of the expanded clusters with reference to the original category labels.

A GA contains many random elements, so we therefore repeat each run 11 times.

3.3.1 Parameters

We used a fixed set of standard GA parameters in all our experiments which are summarised in Table 3. We use an island model with 4 subpopulations as a means to increase diversity and exchange 3 individuals every 30 generations.

Table 3. GA parameters

Parameter	Value
Selection Type	Tournament
Subpopulations	4
Population Size	512
Generations	120
Crossover Probability	0.8
Mutation Probability	0.1

Elitism	Best 2

3.3.2 Multi-word Queries

We can build multiword queries by extending the length of the genome, for example doubling the length of the genome to allow for two-word queries and taking the modulus of *k* to determine which query each gene relates to. A word can only be added once to a set of queries: if the genome specifies two or more occurrences of a particular word, only the first occurrence is used. Where a query is made of two or more words they are connected with a logical OR (disjunction) such that documents are returned which contain any of the words in the query.

When building a query specified by a chromosome, we have found it useful to add a requirement for queries made of two or more words. Each word in a multi-word query can also be used as a single word query. Before we add a new word (*newWord*) to a query already containing a word (*rootWord*), we must first check that the intersect requirement is met by calculating the following:

andCount:	count of documents containing the *newWord* AND the *rootWord*
newWordCount:	count of documents containing the *newWord*
intersectRatio:	*andCount*/*newWordCount*

We have experimented with various values for the minimum *intersectRatio* and have found 0.5 to be a suitable value (see results section below). If *intersect Ratio* >= 0.5 the word is added to the query otherwise nothing is added. In other words, before we add a new word to a query, we check that at least 50% the documents which contain the new word also contain the root word. This method also has the advantage of making the first word in a query more likely to be a good cluster label.

3.4 Example generating 3 search queries (SQ0, SQ1, SQ2)

Table 4 shows and example where k is determined in the first gene of the chromosome and the rest of the chromosome is used to build up a multi-word query.

Table 4: chromosome to determine k and create 3 search queries (SQ)

Representation	**K**	**SQ0**	**SQ1**	**SQ2**	**SQ0**	**SQ1**	**SQ2**	**SQ0**	**SQ1**	**SQ2**
Chromosome	3	0	7	2	3	3	5	1	4	2

In this case the chromosome specifies a *k* value of 3 meaning that 3 clusters will be created. The 3 queries shown in **Table 5** will be created.

Table 5: creating multi-word queries

	Gene	**Specified Words**	**Final Query**	**Comment**
SQ0	0,3,1	*space, orbit, nasa*	*space* OR *orbit* OR *nasa*	*orbit* and *nasa* both have a high intersect ratio with root word *space*
SQ1	7,3,4	*game, orbit, hockey*	*game* OR *hockey*	*orbit* does not meet the intersect requirement for the root word *game* so is not included in the final query.
SQ2	2,5,2	*god, file, god*	*god*	*file* does not meet the intersect requirement for the root word *god*. Repeated word is ignored.

3.5 Fitness Calculation

Text clustering aims to return sets of documents which are related to each other but not related to documents in other clusters. We have created and tested two fitness functions that aim to partition a document collection into clusters by generating a set of search queries. The first fitness function is for the case where the desired number of clusters (*k*) is known in advance. In the second case, the GA will attempt to determine the optimal value for *k*.

When calculating fitness from a set of queries generated by a chromosome, we define *uniqueHits* as the count of documents returned by exactly one query in the set of queries.

Let *Q* be a set of queries, let *D* be a set of documents. Let $M \subseteq Q \times D$ be the set of pairs *(q, d)* where query $q \epsilon Q$ matches document $d \epsilon D$:

$$uniqueHits: |\{d \in D : q \in Q, \exists! (q,d) \in M\}|$$

For the case where *k* is known in advance, we have found that *uniqueHits* is a good fitness measure where the higher the value (the number of documents returned by a exactly one query) the better the fitness.

We have noticed that in the case where *k* is defined in the chromosome the GA often produces solutions with too many categories with respect to the labelled collections. In fact, this is to be expected since overlapping clusters do not lead to a reduction in fitness. We found that introducing a small penalty for more clusters, as in the second fitness test (below), improved effectiveness.

$$uniqueHits * (1 - (k * \text{penalty}))$$

We have found a suitable value for penalty to be 0.02. This value is examined in the results section. Below we show pseudo code to calculate *uniqueHits* for a set of words generated from a chromosome. Note that in Apache Lucene 'SHOULD' is similar to connecting terms with an OR in a query and 'MUST_NOT' is used to indicate that the term MUST NOT occur if a query is to return a document.

```
Set queries = [q0..qk]
int uniqueHits = 0

queries.each {q ->
  BooleanQuery uniqueQ= new BooleanQuery()
  uniqueQ.add(q, SHOULD)
  Set otherQueries = queries.remove(q)

 otherQueries.each {otherQ ->
   uniqueQ.add(otherQ, MUST_NOT)
  }

  uniqueHits += search(uniqueQ).hits.size
}

return uniqueHits
```

3.6 Effectiveness Measures

Effectiveness is determined by referring to the original category labels (ground truth) from the relevant collection.

3.6.1 V measure

We use V-measure [30] as the primary method of assigning effectiveness. The V-measure is based on a combination of homogeneity (*h*) and completeness (*c*). A perfectly homogeneous clustering is one where each cluster has data-points belonging to the same class label. Homogeneity describes the closeness of the clustering algorithm to this perfection.

A perfectly complete clustering is one where all data-points belonging to the same class are clustered into the same cluster. Completeness describes the closeness of the clustering to this perfection. The V-measure score is the harmonic mean of homogeneity and completeness as given by

$$V = \frac{(1+\beta) * h * c}{(\beta * h + c)}$$

We assign a default value of 1 to beta so that homogeneity and completeness are given equal weighting.

3.6.2 Adjusted Rand Index

We also provide the adjusted Rand Index (ARI) [9] as a secondary performance measure. ARI is a measure of the similarity between the clusters produced by the algorithm and the original document labels. The ARI is calculated as follows:

$$ARI = \frac{2(agreement - chance)}{agreement + chance}$$

where:

- agreement is the number of pairs of points that are assigned to the same cluster in both clusters.
- chance is the expected number of pairs of points that would be assigned to the same cluster by chance, given the number of clusters and the size of the data set.

The ARI can take values between -1 and 1, where -1 indicates perfect disagreement and 1 indicates perfect agreement. A value of 0 indicates that the two clusters are no better than random.

3.6.3 Cluster count error

We also provide the cluster count error which is simply the absolute value of the number of classes minus the number of clusters. This measure is only relevant for the case where *k* is not known in advance.

3.7 Definitions

Let W be the set of all words in any document in a collection, so $W \subseteq \wp(\Sigma^*)$, here $\wp(X)$ is the power set of X, Σ is a finite alphabet and Σ^* is the set of finite strings over Σ.

We consider a document as an unstructured set of words, i.e. a document belongs to $\wp(W)$. In this way, we ignore the order and multiplicity of the words in the document. [Later, we may consider a document as a multi set of words.]. Let $D \subseteq \wp(\wp(W))$ be a set of documents.

In the simplest case, a query is a single word $w \in W$. By membership, each query defines the set $\delta(w) \subseteq D$, of all documents d such that $w \in d$. More generally, a query q is a set of words (i.e. $q \in \wp(W))$. The query q matches the document $d \in D$ if and only if $q \cap d \neq \emptyset$ i.e. if at least one word of the query occurs at least once in the document. Again, for any query q we define $\delta'(q) \subseteq D$ to be the set of documents d such that q matches d. Observe that

$$\delta'(q) = \bigcup_{w \in q} \delta(w)$$

A chromosome $(q_i{:}i < k)$ is a sequence of k queries (some $k > 1$). A *uniqueHit* for a chromosome occurs when a document matches exactly one of its k queries. The *uniqueHitcount* for a chromosome is the count of documents matching exactly 1 query in the set. Let

$$u(q_i{:}i < k = \bigcup_{i<k} \delta'(q_i) \setminus \bigcup_{i<j,k} (\delta'(q_i) \cap \delta'(q_j))$$

The symmetric difference of the $\delta'(q_i)s$, i.e., the set of all documents that are matched by exactly one of the k queries.

A *class labelling* of D is any finite partition of D, so a class labelling $\{S_i : i < s\}$ consists of s disjoint non-empty sets (some finite s) whose union is the whole of D. s is the size of the partition. So, a class labelling belongs to

$\wp(\wp(D))$.

Let $C = \{S_i : i < s\}$ be a class labelling of size s, and let $K = \{q_j : j < k\}$ be a set of k queries for some finite k. For $j < k$ let $Q_j = \delta'(q_j) \subseteq D$, the set of documents that match q_j.

We may define the V measure of (*C, K*) by

$$V(C,K) = \frac{2hc}{h+c}$$

where

$$h = 1 - \frac{H(C|K)}{H(C)}$$

$$c = 1 - \frac{H(K|C)}{H(K)}$$

$$H(C \mid K) = -\sum_{i<s,j<k} \left(\frac{|S_i \cap Q_j|}{D} .log\left(\frac{|S_i \cap Q_j|}{|Q_j|} \right) \right)$$

$$H(K \mid C) = -\sum_{i<s,j<k} \left(\frac{|S_i \cap Q_j|}{D} .log\left(\frac{|S_i \cap Q_j|}{|S_i|} \right) \right)$$

$$H(C) = -\sum_{i<s} \left(\frac{\Sigma_{j<k} |S_i \cap Q_j|}{s} .log\left(\frac{\Sigma_{j<k} |S_i \cap Q_j|}{s} \right) \right)$$

$$H(K) = -\sum_{j<k} \left(\frac{\Sigma_{i<s} |S_i \cap Q_j|}{s} .log\left(\frac{\Sigma_{i<s} |S_i \cap Q_j|}{k} \right) \right)$$

3.8 Cluster Expansion using KNN

The clusters produced by the GA generated search queries have a drawback in that many of the documents are not returned by any query; on average only 70% of the documents are clustered. If we discard the documents which are not in any cluster and then analyse the remaining documents which are clustered with reference to the original class labels the clusters have a high V-measure, mostly above 0.9 and sometimes approaching 1. However, it is usually the case that we need to add every document to a cluster. To achieve this, we include a second stage whereby the query generated clusters are used as training documents for a classifier. We use a KNN classifier to assign each of the unassigned documents to their nearest cluster. Figure 1 shows the NG3 collection where an X represents a document which has been assigned to a category. For example, cluster A shows all the documents which contain the word '*god*'. Y indicates a document which does not contain any of the search query words ('*god*', '*hockey*' or '*nasa*') and are therefore not included in any cluster. The arrows represent the process whereby the Lucene implementation of KNN assigns documents y0 – y3 to the nearest cluster. We use a Euclidean distance measure with a K value of 10.

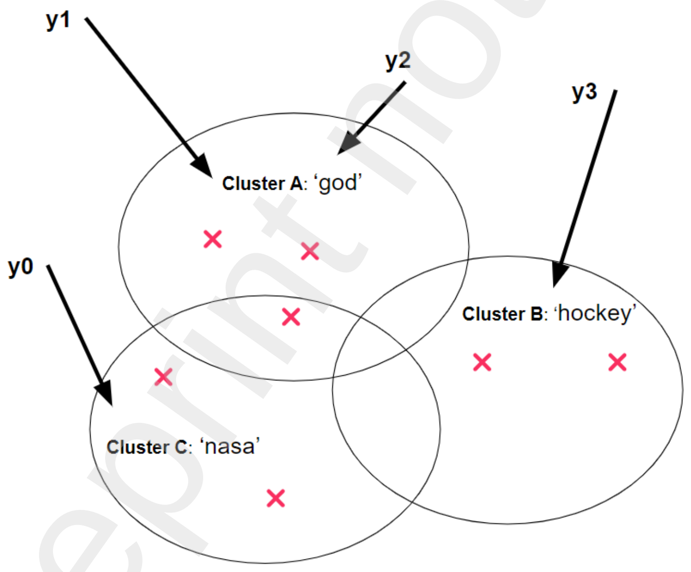

Figure 1: KNN cluster expansion

4 Results and Discussion

4.1 Intersect requirement

The intersect requirement was developed to support multi word query building in the situation where *k* is not known in advance. In this situation, if the intersect constraint is not included the GA will almost always select a value of 2 for the number of clusters (*k*). For example, a typical clustering for the NG5 set is shown by the set of 2 queries is shown in Table 6

Table 6. Clusters generated with no intersect requirement.

Cluster	Query Words	Document Hits
1	hockey nhl game players	208
2	sale please mail windows god space high work apr anyone	1357

The fitness test is a based on the count of documents returned by exactly one query, so the query set shown achieves a high fitness, but the number of clusters (2) does not match the number of labelled classes (5) and the second query is returning documents from multiple classes. Completeness is high (0.852) but homogeneity low (0.212) and the V-measure for this clustering is also low (0.340).

We can improve things by restricting the GA to using one word per cluster. In this case, using more queries can result in more unique hits and higher fitness. A typical result is shown with the set of queries below:

Table 7. single word queries (NG5)

	Query Word	**Document hits**
1	space	204
2	windows	278
3	team	176
4	sale	192
5	god	205

The correct number of categories has been identified, and the evaluation metrics show a distinct improvement (v: 0.773, h: 0.773, c: 0.774).

The intersect constraint allows an individual to add more words to a single term query, but only when the set of documents retrieved by the first term (root term) in a query intersects with the set of documents retrieved by any new term added to the query. In the results shown below we require that 50% of the documents retrieved by the new term are found in the set retrieved by the root term. The rationale for the intersect requirement is to create a mechanism which allows GAs to produce queries with multiple terms, but only retrieving related documents ideally from a single category. Using the intersect requirement we get an even bigger improvement. A typical result of a run using the intersect constraint is shown in Table 8.

Table 8. multi-word queries with intersect requirement (NG5)

	Query words	Document Hits
1	sale	192
2	windows files	295
3	game players hockey games	299
4	god christ jesus church	294
5	space moon nasa	262

The GA can add more keywords to queries provided the intersect requirement is met for each new term. The set of queries above has correctly created 5 clusters with (v: 0.882, h: 0.880, c: 0.883).

We ran the GA across all the indexes with various values between 0 and 1 for the intersect requirements and present the results in Figure 2.

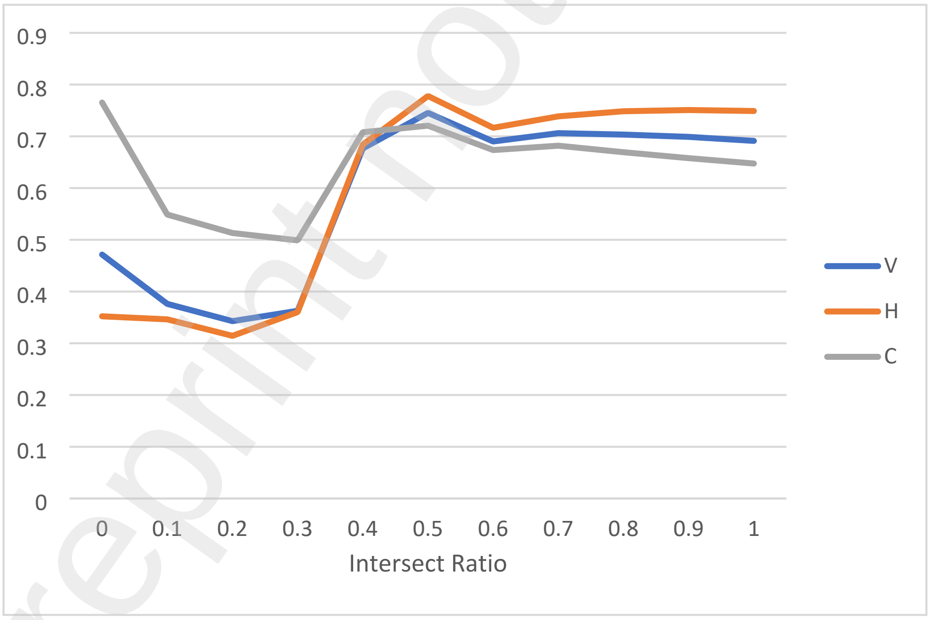

Figure 2: Average V, H and C scores across all 8 datasets for different values of the minimum intersect ratio where k is discovered, and multi-word queries are enabled.

Following these results, we use an intersect ratio of 0.5 in the experiments described below.

4.2 Penalty for more clusters

In the situation where *k* is not known in advance, the number of clusters produced by the GA is typically higher than the number of categories existing in the original collection. This higher fragmentation leads to weaker results, and we found effectiveness could be improved by introducing in the fitness a small penalty based on the number of clusters:

```
fitness = uniqueHits * (1.0 - (kPenalty * k))
```

We investigated various values for the penalty (k penalty) as shown in Figure 3

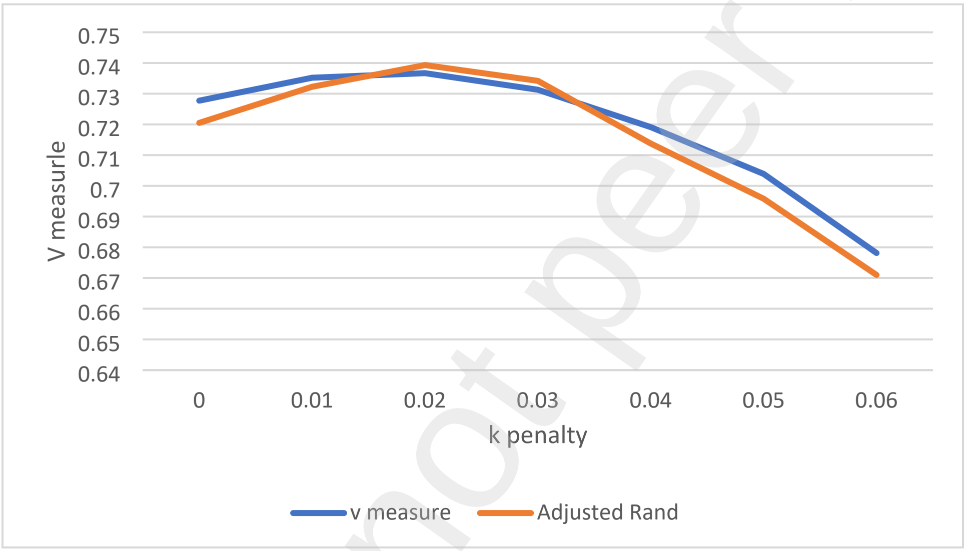

Figure 3: average values for V and Adjusted Rand Index across all 8 datasets for different values of k penalty where k is discovered, and multi-word queries are enabled.

We also investigated how the cluster count error responds to different values of *k* penalty as shown in Figure 4.

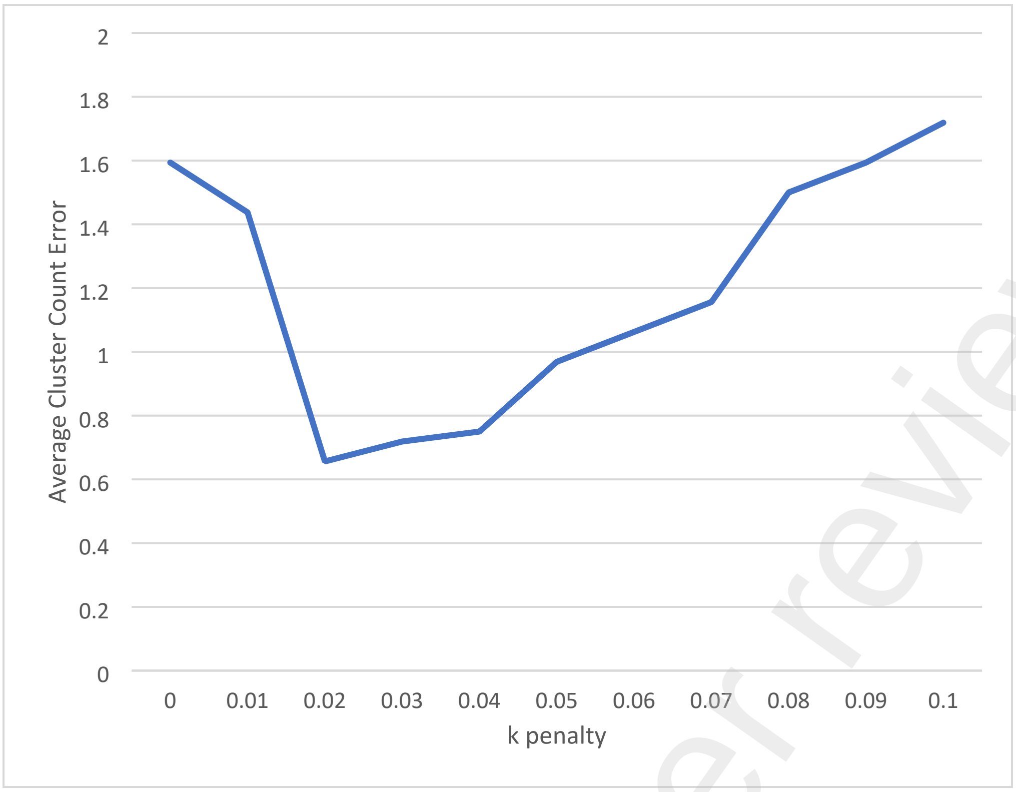

Figure 4: average cluster count error across all 8 datasets for different values of k penalty

V measure and Adjusted Rand peak with a *k* penalty of 0.02 and the cluster count error is at its lowest for this value. Following these results, we selected a penalty of 0.02 in the experiments described below for the case where *k* is not known in advance.

4.3 Overview

Table 9: average across all indexes

	Query Type	**v-measure**	**ARI**
k-discovered	**Multi-word**	0.730	0.728
	Single-word	0.677	0.650
k-predefined	**Multi-word**	0.731	0.749
	Single-word	0.697	0.710

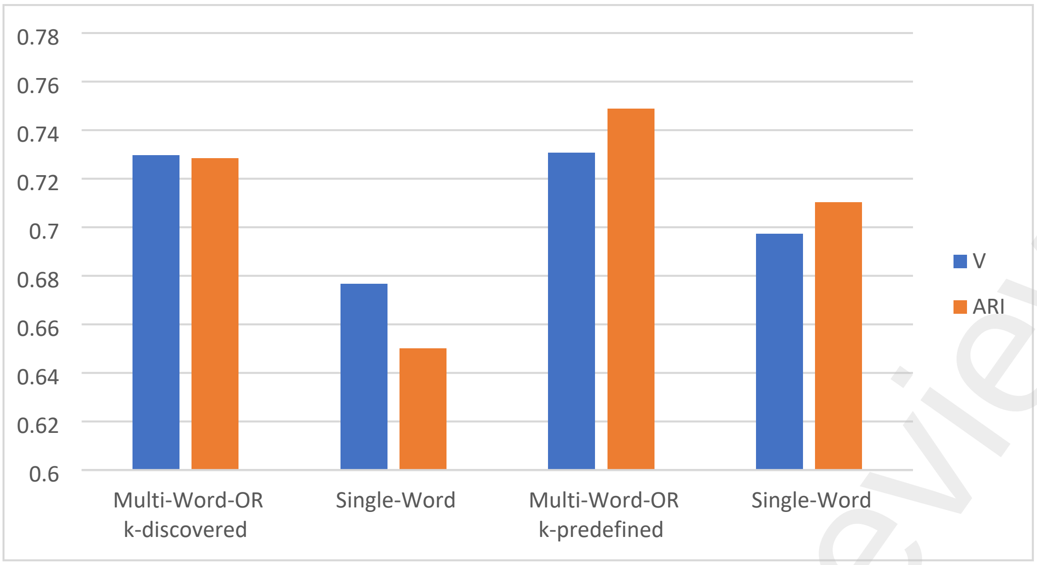

Figure 5: results overview

Multi-word queries perform better than single word queries. Where *k* is given in advance results are slightly improved.

4.4 Comparison with k-means ++

We present a basic comparison across the 8 datasets for the eSQ (multi-word) system and the implementation of k-means++ in scikit-learn [9]. The value of *k* is given in advance for k-means++ but we show the results for eSQ where *k* is discovered (a harder problem). 11 runs were obtained for both systems and the average value of the V measure and ARI is shown. We use a tf-idf based vectorizer and a feature size of 1000 for k-means++.

Table 10: comparison of eSQ and k-means++

	v		ARI	
	eSQ	**k-means++**	**eSQ**	**k-means++**
CRISIS3	0.600	0.451	0.580	0.322
CRISIS4	0.682	0.565	0.673	0.421
NG3	0.910	0.865	0.944	0.9
NG5	0.731	0.718	0.734	0.632
NG6	0.733	0.657	0.744	0.524
R4	0.841	0.632	0.864	0.5
R5	0.709	0.529	0.720	0.444
R6	0.633	0.607	0.568	0.459
Average	0.730	0.628	0.728	0.525

These results are visualized in Figure 6 and Figure 7 showing that eSQ (*k* discovered) outperforms k-means++ in every dataset.

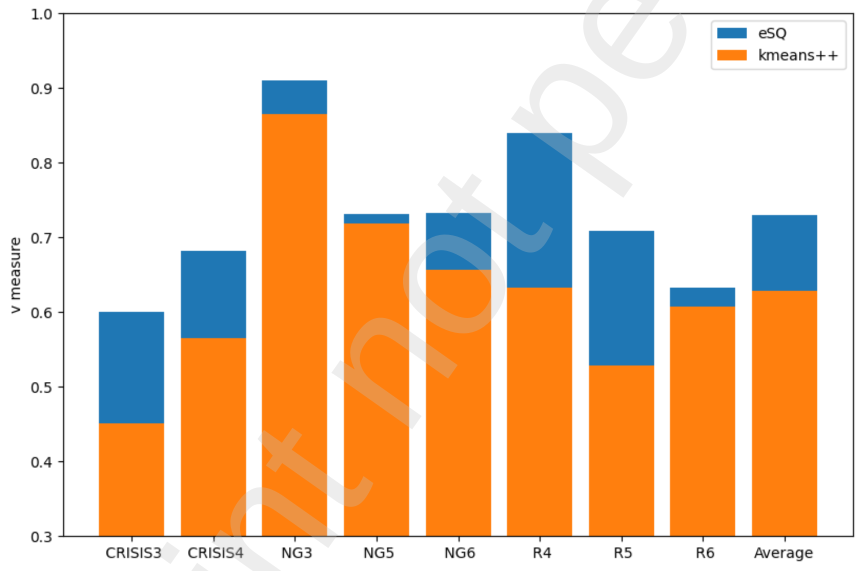

Figure 6: V measure for eSQ (k discovered) and k-means++

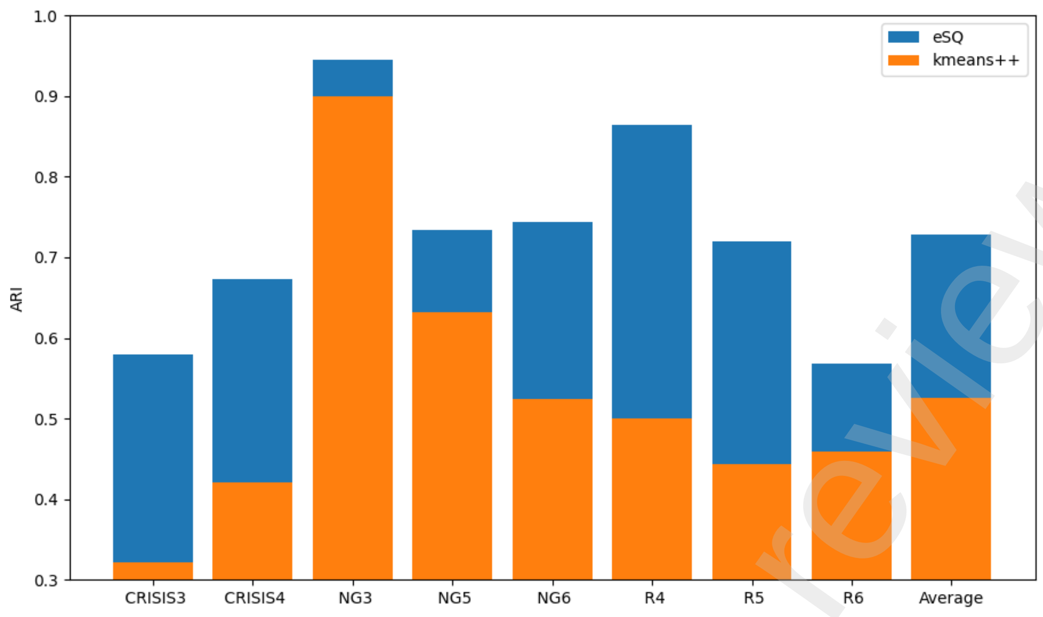

Figure 7: ARI for eSQ (k discovered) and k-means++

Table 11 shows the standard deviation across the 11 runs.

Table 11. Standard deviation of V measure

	eSQ	**k-means ++**
CRISIS3	0.007	0.121
CRISIS4	0.005	0.081
NG3	0.002	0.025
NG5	0.009	0.070
NG6	0.005	0.041
R4	0.028	0.071
R5	0.012	0.062
R6	0.029	0.081
Average	0.099	0.138

4.5 Comparison with agglomerative clustering and spectral clustering

We also tested using agglomerative clustering and spectral clustering [31], but both performed quite poorly compared to k-means++ or eSQ. Agglomerative clustering also has the advantage of not requiring the number of clusters to be provided in advance.

Table 12 V measure for Agglomerative and Spectral clustering

	Agglomerative	Spectral
crisis3	0.243	0.040
crisis4	0.293	0.067
NG3	0.309	0.813
NG5	0.364	0.610
NG6	0.403	0.639
R4	0.446	0.433
R5	0.474	0.302
R6	0.467	0.326
Average	0.375	0.404

Spectral clustering is often performing better than agglomerative clustering but is failing to effectively cluster the short text (tweet) data in the crisis datasets.

5 Conclusions and future work

We have presented eSQ, a novel system for text clustering which is based on GA generated search queries. The eSQ system can produce effective text clustering by using search queries at cluster centres, with the advantage that the exact number of clusters does not need to be provided in advance. eSQ is different to most modern clustering systems which would use a document or a point in a multi-dimensional space as the cluster centre. The initial clusters (formed from the GA generated search queries) are produced without using a measure of similarity between documents. The search query method has the advantage of interpretability where the search terms function as cluster labels. As mentioned in the introduction the cluster hypothesis suggests that search query method will naturally align with information retrieval requirements [8].

Currently the algorithm only generates disjunctive queries and is only suitable for clustering text documents. We would like to experiment with more complex queries which include conjunction, negation and other search query types as have successfully been used in text classification [23]. We would also like to develop the algorithm so that it could be applied to cluster other media such as images.

References

[1] G. Salton and C. Buckley, “Term-weighting approaches in automatic text retrieval,” *Information Processing and Management,* vol. 24, no. 5, pp. 513-523, 1988.

[2] W. Samek, T. Wiegand and K. Müller, “Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models,” *ITU Journal: ICT Discoveries, Special Issue The Impact of AI on Communication Networks and Services,* vol. 1, pp. 1-10, 2017.

[3] G. Riccardo, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti and D. Pedreschi, “A survey of methods for explaining black box models,” *ACM computing surveys (CSUR),* vol. 51, no. 5, 2018.

[4] T. Wei, Y. Lu, H. Chang, Q. Zhou and X. Bao, “A semantic approach for text clustering using WordNet and lexical chains.,” *Expert Systems with Applications,,* vol. 42, no. 4, pp. 2264-2275., 2015.

[5] E. K. Jasila, N. Saleena and K. A. Abdul Nazeer, “Ontology Based Document Clustering - An Efficient Hybrid Approach,” in *IEEE 9th International conference on advanced computing((IACC)*, Tiruchirapalli, 2019.

[6] N. Allahverdi , H. Kahramanli and M. Koklu , “Rule extraction from linear support vector machines,” in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, 2005.

[7] F. Alam, M. Malik and M. Krishnamurthy, “NeuroClustr: Empowering Biomedical Text Clustering with Neuro-Symbolic Intelligence,” in *International Joint Conference on Artificial Intelligence 2023 Workshop on Knowledge-Based Compositional Generalization*, 2023.

[8] C. D. Manning, R. Raghavan and H. Schultze, Introduction to Infromation Retrieval, Cambridge University Press, 2008.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* vol. 12, pp. 2825--2830, 2011.

[10] D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding.,” in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.

[11] B. Radomirović, V. Jovanović, B. Nikolić, S. Stojanović, K. Venkatachalam, M. Zivkovic and I. Strumberger, “Text Document Clustering Approach by Improved Sine Cosine Algorithm,” *Information Technology and Control,* vol. 52, no. 2, pp. 541-561, 2023.

[12] X. Han, Y. Zhu, K. M. Ting and G. Li, “The impact of isolation kernel on agglomerative hierarchical clustering algorithms,” *Pattern recongition,* vol. 139, 2023.

[13] D. Merlini and M. Rossini, “Text categorization with WEKA: A survey,” *Machine Learning with Applications,* vol. 4, 2021.

[14] Y. Rong and Y. Liu, “Staged text clustering algorithm based on K-means,” in *IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2020.

[15] J. H. Holland, “Genetic Algorithms,” *Scientific American,* vol. 267, no. 1, pp. 66-72, 1992.

[16] E. R. Hruschka, R. J. Campello and A. A. Freitas, “A survey of evolutionary algorithms for clustering,” *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 39, no. 2, pp. 133-155, 2009.

[17] A. K. Uysal and S. Gunal, , “Text classification using genetic algorithm oriented latent semantic features,” *Expert Systems with Applications, 41(13),* pp. 5938-5947., 2014.

[18] W. Song, Y. Qiao, S. C. Park and X. Qian, “A hybrid evolutionary computation approach with its application for optimizing text document clustering,” *Expert Systems with Applications,* vol. 42, no. 5, pp. 2517-2524, 2015.

[19] H. J. Escalante , M. A. García-Limón, A. Morales-Reyes, M. Montes-y-Gómez and A. Martínez-Carranza, “Term-weighting learning via genetic programming for text classification,” *Knowledge-Based Systems, 83,* pp. 176-189, 2015.

[20] D. Mustafi, A. Mustafi and G. Sahoo, “A novel approach to text clustering using genetic algorithm based on the nearest neighbour heuristic,” *International Journal of Computers and Applications,* vol. 44, no. 3, pp. 291-303, 2022.

[21] C. Clack, J. Farringdon, P. Lidwell and T. Yu, “ Autonomous document classification for business,” in *Proceedings of the first international conference on Autonomous agents* , 1997.

[22] A. Pietramala , V. Policicchio , P. Rullo and I. Sidhu, “A Genetic Algorithm for Text Classification Rule Induction,” in *Proc. European Conf. Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD ’08)*, 2008.

[23] L. Hirsch and T. Brunsdon, “A comparison of Lucene search queries evolved as text classifiers,” *Applied Artificial Intelligence,* vol. 32, no. 7, pp. 768-784., 2018.

[24] L. Hirsch and A. Di Nuovo, “Document Clustering with Evolved Search Queries,” in *Evolutionary Computation (CEC), IEEE Congress on.*, Donostia - San Sebastián, 2017.

[25] L. Hirsch, A. Di Nuovo and P. Haddela, “Document clustering with evolved single word search queries,” in *IEEE Congress on Evolutionary Computation (CEC)*, 2021.

[26] C. Jia, M. B. Carson, X. Wang and J. Yu, “Concept decompositions for short text clustering by identifying word communities,” *Pattern Recognition,* vol. 76, pp. 691-703, 2018.

[27] E. Maden and K. Pinar, “Recent methods on short text stream clustering: A survey study,” *Wiley Interdisciplinary Reviews: Computational Statistics:,* 2023.

[28] A. Olteanu, S. Vieweg and C. Castillo, “What to expect when the unexpected happens: Social media communications across crises,” in *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, 2015.

[29] K. Lang, “Newsweeder: Learning to filter netnews,” in *Proceedings of the Twelfth International Conference on Machine Learning.*, 1995.

[30] A. Rosenberg and J. Hirschberg, “ V-measure: A conditional entropy-based external cluster evaluation measure,” in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007.

[31] M. Filippone, F. Camastra, F. Masulli and S. Rovetta, “A survey of kernel and spectral methods for clustering,” *Pattern recognition,* vol. 41, no. 1, pp. 176-190, 2008.