International Journal on Artificial Intelligence Tools

Improving Hierarchical Short Text Clustering through Dominant Feature Learning

Leonidas Akritidis
Department of Science and Technology, International Hellenic University
Thessaloniki, Greece
lakritidis@ihu.gr

Miltiadis Alamaniotis
Department of Electrical and Computer Engineering, University of Texas at San Antonio
San Antonio, USA
Miltos.Alamaniotis@utsa.edu

Athanasios Fevgas, Panagiota Tsompanopoulou
Department of Electrical and Computer Engineering, University of Thessaly
Volos, Greece
{*fevgas,yota*}*@e-ce.uth.gr*

Panayiotis Bozanis
Department of Science and Technology, International Hellenic University
Thessaloniki, Greece
pbozanis@ihu.gr

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

This paper focuses on the popular problem of short text clustering. Since the short text documents typically exhibit high degrees of data sparseness and dimensionality, the problem in question is generally considered more challenging than the traditional clustering scenarios. Our proposed solution, named VEPH, is based on a novel algorithm that was published recently with the aim of optimally clustering short text documents. VEPH includes two stages: During the first stage, the original text vectors are projected on a lower dimensional space and the documents with projection vectors lying on the same dimensional space are grouped in the same cluster. The second stage is a refinement process which attempts to improve the quality of the clusters that were generated during the previous stage. The quality of a cluster is determined by its homogeneity and completeness and these are the two primary design criteria of this stage. Initially VEPH cleanses the clusters by removing all dissimilar elements, and then, it iteratively merges the similar clusters in a hierarchical agglomerative manner. The proposed algorithm has been experimentally evaluated in terms of F1 and NMI, by employing three datasets with diverse attributes. The results demonstrated its superiority over other state-of-the-art works of the relevant literature.

Keywords: clustering; short text clustering; machine learning; unsupervised learning.

1. Introduction

The problem of short text clustering has emerged during the last decade due to the explosive growth of applications that base their operation on the generation, processing and dissemination of short textual information. Representative examples include microblogs, instant messengers, mail clients, Q&A communities, search engines, platforms that handle entitled entities (such as news portals and article aggregators, product comparison services, e-commerce applications, etc.), and so on. Since the identification of semantically similar entities is of crucial significance for the majority of these services, it follows that short text clustering constitutes a particularly hot problem these days.

In general, data clustering is an extensively studied unsupervised learning problem that requires the recognition of similar entities (also called data points) from a set of unlabelled data, and the subsequent grouping of them into a set of clusters. In this context, the two main goals of a clustering algorithm are: i) to group *all* similar elements into the same cluster, and ii) to place *all* dissimilar elements into different clusters. The first goal is connected to the notion of *cluster completeness* and expresses the ability of the algorithm to group as many similar items as possible inside the same cluster. On the other hand, the second goal reflects *cluster homogeneity*, that is, the integration of no, or few dissimilar items within a cluster.

The current state-of-the-art approaches have identified *data sparseness* and *high dimensionality* as two of the most significant problems that discriminate short text clustering from traditional data clustering. Their existence renders the problem considerably more challenging than normal clustering,[1–4] because both of them have a negative impact in the performance of the standard data clustering algorithms.

More specifically, short texts exhibit a higher degree of sparseness compared to the normal text representations. This happens because the probability that two short documents share many important words is small, or, at least, it is smaller than the respective probability in the case of normal documents. Other reasons concern synonymy and polysemy. The first works in the area attempted to limit the effects of sparseness by augmenting the input document collections with information originating from external Web sources.[5,6] Nevertheless, the prohibitively high cost of these methods motivated the research community to abandon this approach and introduce corpus-based and self-taught methods.[3,7] Some recent state-of-the-art works proposed techniques for substituting specific words by their synonyms,[8] algorithms for identifying the latent relationships between pairs,[9,10] or combinations of terms,[4,11,12] and self-taught neural networks that learn non-biased deep text representations.[13]

Regarding the problem of high dimensionality, the standard reduction algorithms can be also applied in the case of short text clustering. Although they adopt different strategies, the goal of the vast majority of them is common: to project the input feature vectors into a lower dimensional space. Hence, SVD, PCA, QR, and eigen decomposition (and their variants) limit the size of the original term-document

matrix via factorization. Nevertheless, these approaches have two major drawbacks: First, they employ matrices to represent the input documents, thus they do not scale well with the size of the dataset. And second, they require prior knowledge of the true number of clusters; this requirement renders them infeasible in cases where the input is either unknown, or changes rapidly.

In this paper we present VEPH, an algorithm for clustering short documents and entitled entities. VEPH is an extension of VEPHC which was recently introduced with the aim of limiting the sparseness of short texts, while it simultaneously improves the completeness and homogeneity of the generated clusters.[4] Similarly to its predecessor, VEPH also includes two stages: The first one performs an initial clustering of all similar items in the dataset. It achieves this goal by initially identifying the best combination of the features of a short document, according to a scoring function. Then, it treats this combination as a projection of the original feature vector onto a lower dimensional space, and groups all the entries with common projection vectors into the same cluster.

The second stage includes an effective refinement algorithm which improves the initial clustering of the previous stage. In VEPHC, this was implemented in a cluster split-merge fashion. In VEPH, the split phase is almost identical. However, the merging of the proximal cluster takes place in a hierarchical agglomerative manner. Notice that in this refinement stage, the split phase aims to improve the homogeneity, whereas the merge phase is designed to enhance the completeness of the generated clusters.

This work introduces several novel features summarized into the following list:

- In most machine learning tasks, a text dataset is typically vectorized according to the well-known tf-idf method. However, when applied to short texts, the term frequency is almost always equal to 1, therefore, the feature weights are only affected by the term's inverse document frequency (IDF). To address this problem, we propose a different strategy for creating text vectors that takes into consideration the offset of a word in the short text,
- VEPH is based on a novel dimensionality reduction approach that is not related to the typical matrix factorization and/or matrix decomposition methods of the relevant literature. Instead, it forms combinations of features and identifies the best (termed as "dominant") among them. Then, the dominant feature combinations are treated as projections of the initial vectors and the documents with common dominant projections are eventually grouped inside the same cluster,
- The idea of creating combinations of features and finding the dominant feature combination is made more solid with the introduction of a more robust scoring function,
- Our method introduces a cluster refinement stage that initially removes all dissimilar elements from the initial clusters. Then, in contrast to VEPHC, it iteratively merges all similar (or proximal) clusters in a hierarchical fashion.

The rest of this article is organized as follows: Section 2 briefly presents the most inspiring works in the research area of short text clustering. A formal description of the problem along with the necessary background knowledge are provided in Section 3. The core elements and the main contributions of the article are subsequently presented in a detailed manner in Sections 4 and 5. Furthermore, in Section 6 we report the results of the experimental evaluation of the proposed algorithm, accompanied by a thorough interpretation. Finally, the highlights and the conclusions of this article are summarized in Section 7.

2. Related Work

Text clustering is among the most popular problems in the wide area of unsupervised learning and the current literature includes a wide variety of relevant state-of-the-art studies. Besides, the broad adoption of short text documents in numerous applications along with their native sparseness, rendered short text clustering a very attractive field of research.

One of the first contributions in the area was Spherical k-means,[1] a variant of the well-known k-means algorithm which attempted to address the weakness of its predecessor in handling the sparseness of short texts. More specifically, the algorithm introduced a decomposition method for constructing concept vectors from the cluster centroids, with the aim of achieving effective clustering of normal texts. In contrast, the decomposition approach of Jia et al.[2] created a weighted term co-occurrence graph and exploited these weights to identify word communities with similar semantics.

Moreover, several concept and matrix factorization methods[14,15] were introduced in order to extend the aforementioned concept decomposition algorithms. One of the most successful factorization techniques is the well-established Non-negative Matrix Factorization[16] (NMF). To address the problem of sparseness of short documents, Yan et al.[17] employed NMF in combination with an exploration strategy of the word correlation data. Moreover, a novel word weighting method for NMF was proposed by the same group.[18] Their approach is based on the normalized cut problem when it is applied on the term affinity matrix.

Although NMF has been proved effective in multiple clustering tasks, it takes into consideration only the global Euclidean space to compute new basis vectors, thus ignoring the local manifold geometry of the underlying data. For this reason, Shang et al.[19] proposed a graph-based dual regularization (DNMF) extension, which exploits the geometric properties of both the data and feature manifolds. Similarly, CGF also constitutes an effective dual-graph regularized concept factorization method for text clustering.[20]

Another branch of relevant works focused on the identification of individual topics inside text corpora through the utilization of probabilistic models. Latent Dirichlet Allocation[21] (LDA) is one of the most effective algorithms of this family. LDA was extensively studied in many topic modelling problems including short text

clustering.[22] In this context, a recent model named GSDMM introduced an iterative Gibbs Sampling algorithm for the Dirichlet Multinomial Mixture.[23] GSDMM does not require setting the number of clusters in advance, while it achieves good clustering performance in several different tasks. Nevertheless, since it is based on an iterative sampling algorithm, it is rather inappropriate for handling large datasets with numerous clusters.

In addition, Yan et al.[9] proposed BTM, a method for capturing and modelling the observed word co-occurrence patterns in a document collection. Unfortunately, the word-occurrence in BTM is limited to pairs of words (namely, biterms), and ignores the possible relationships among three or more terms. In contrast, VEPH implements an extended word-occurrence logic by creating combinations of multiple words. Consequently, it is able to discover more qualitative information about the correlations of short texts.

In many unsupervised learning tasks, the concept of self-teaching has led to powerful algorithms with increased performance. Following this trend, the recent state-of-the-art self-taught short clustering methods introduced models for learning features, relationships, constraints, etc. The well-established convolutional neural networks provided several promising models for learning deep text representations.[13,24,25] Alternative robust solutions can also derive from the usage of autoencoders.[26,27] Notice that our proposed method includes a refinement stage which can also be considered as a self-taught procedure.

In the sequel, let us refer to some methods that purely attempt to overcome the sparseness of short text documents. The vast majority of them artificially enrich the representations of short texts with extra features (that is, words, terms, phrases, etc.). The first approaches employed external data sources such as search engines,[5] Wikipedia,[6] HowNet,[28] and WordNet[29] with the aim of fetching relevant information. Nonetheless, all of them suffered from serious efficiency problems originating from the underlying communication and post-processing overheads.[30]

For this reason, a second family of self-taught methods, also known as corpus-based methods,[8] emerged with the aim of achieving the enrichment goal without utilizing external information sources. For example, Pinto et al.[7] proposed a methodology which constructed knowledge bases after processing the same input dataset. Then, an algorithm exploited these bases for inserting co-related terms to a short document. Another similar corpus-based method was introduced by Zheng et al.[3] who suggested a mapping mechanism for projecting the original documents onto a hidden semantic space.

Finally, there is also a number of systematic overviews of the relevant literature. In one of them, Rangrej et al.[31]conducted a comparative study of the most important short text clustering algorithms before 2011. A more recent survey on topic modelling for short text documents was published by Likhitha et al.[32]

3. Preliminary Elements, Notation, and Dataset Vectorization

This section contains a brief presentation of the basic elements that will provide the theoretical foundations for the proposed method. It also introduces a new technique for transforming short text documents into vectors; some works refer to this transformation as text vectorization.

We begin with the introduction of an input document collection D, consisting of n records (also called samples). In the particular case that we examine, each sample d_i contains a small number of terms and is typically represented by a dense vector[a] $\mathbf{x}_i = \{x_{i1}, \ldots, x_{il_i}\}$. In this representation, x_{ij} denotes the weight of the jth term of d_i, whereas l_i is the length of d_i in number of terms. Equivalently, l_i also represents the number of elements of $\mathbf{x}_i$, hence, it essentially determines its dimensionality.

Regarding the vector coefficients x_{ij}, the *tf-idf* vectorization method is a quite popular strategy to determine their value:

$$x_{ij} = tf_{ij} \cdot \text{IDF}(t_j) = tf_{ij} \cdot \log(n/f_j), \tag{1}$$

where tf_{ij} denotes the number of the occurrences of the term t_j in the document d_i, whereas f_j is the frequency of t_j in the entire document collection. This weighting method has been proved effective in many text mining and IR applications. However, the small length of short texts minimizes the probability for a term to occur multiple times in a document. So for the vast majority of the terms, it holds that $tf = 1$ and the scores of Eq. 1 are primarily determined by the inverse document frequency (IDF) of a term.[2]

Moreover, several experimental studies demonstrated that the position of a term in a document is occasionally an indication of its informational value. Equivalently, the important words tend to appear early in the text. Motivated by these observations and the ineffectiveness of *tf-idf* in applications involving short texts, we introduce *tp-idf*, a technique that incorporates the position of a term and its inverse document frequency according to the following equation:

$$x_{ij} = \text{IDF}(t_j) \sum_{\forall t_j \in d_i} \frac{1}{\alpha + \log(p_{ij} + 1)} = \log \frac{n}{f_j} \sum_{\forall t_j \in d_i} \frac{1}{\alpha + \log(p_{ij} + 1)} \tag{2}$$

where p_{ij} is the position of the jth term of d_i in the document d_i, and α is a real hyper parameter that falls into the range $alpha \in [0, 1]$. Notice that Eq. 2 takes into consideration the potential multiple occurrences of a term in a document and assigns weights according to its position(s). By applying the proposed *tp-idf* weighting function, the document collection D is eventually transformed into a vector space $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

[a] A short text document can also be represented by a sparse vector $\mathbf{x}_i^s = \{x_{i1}, \ldots, x_{i|W|}\}$, where $|W|$ is the total number of terms in the collection. In a sparse vector, the vast majority of its components are zero, leading to space ineffective representations. In this paper we exclusively focus on dense vector representations.

Table 1. Notation.

Symbol	Meaning
D	The input dataset, or corpus
$d \in D$	A short text document in D
n	The number of documents in the corpus
$\mathbf{x}$	A vector representation of d
$l_d, l_{\mathbf{x}}$	The lengths of d and $\mathbf{x}$; it holds that $l_d = l_{\mathbf{x}}$
x_j	The jth element of $\mathbf{x}$
$\mathbf{x}'$	A projection of $\mathbf{x}$ on another feature space
$\hat{\mathbf{x}}'$	A unit vector with the same components as $\mathbf{x}'$ (Reference Vector of $\mathbf{x}$)
$\mathbf{x}^*$	The dominant (highest scoring) projection vector of $\mathbf{x}$
$\hat{\mathbf{x}}^*$	A unit vector with the same components as $\mathbf{x}^*$ (Dominant Reference Vector of $\mathbf{x}$)
$\mathbf{X}$	A feature space
k	The number of the generated clusters
C	The cluster universe
$c \in C$	A cluster cosisitng of similar short text documents

Example 1. Consider a document $d = \{t_1, t_2, t_1\}$ with $\text{IDF}(t_1) = 1$ and $\text{IDF}(t_2) = 2$. According to the proposed *tp-idf* method (Eq. 2), and by setting $\alpha = 1$, we have:

$$x_1 = 1 \cdot \Big(\frac{1}{1+\log(1+1)} + \frac{1}{1+\log(3+1)}\Big) \simeq 0.769 + 0.624 = 1.393, \quad \text{and}$$

$$x_2 = 2 \cdot \frac{1}{1+\log(2+1)} \simeq 1.354$$

Consequently, d is represented by the vector $\mathbf{x} = (1.393, 1.354)$.

For simplicity reasons, and without loss of generality, we drop the subscript i for the rest of this presentation. VEPH is based on the projection of the original text vectors $\mathbf{x}$ onto a lower dimensional space $\mathbf{X}'$. The advantages of this approach will be analyzed in the next section. For the time being, in this article we shall use the notation $\mathbf{x}'$ to symbolize a projection of $\mathbf{x}$ on the space $\mathbf{X}'$. Moreover, $\hat{\mathbf{x}}'$ is the *reference vector (RV)* of $\mathbf{x}$, that is, a unit vector having the same elements as the projection vector $\mathbf{x}'$, and length equal to 1.

Section 4 will introduce a special function to assign scores $S_{\mathbf{x}'}$ to each projection $\mathbf{x}'$ of $\mathbf{x}$. Let $\mathbf{x}^*$ be the *dominant projection vector (DPV)* of $\mathbf{x}$, namely, the projection of $\mathbf{x}$ which achieved the highest score. Similarly, we also define the *dominant reference vector (DRV)* of $\mathbf{x}$ as a unit vector with the same components as $\mathbf{x}^*$.

Now the goal of a clustering algorithm is to appropriately group the short text documents of D into a set C consisting of k clusters. C is also called as the *cluster universe* and a member of C is merely a cluster c. Table 1 conveniently summarizes all the aforementioned notation.

4. Identifying Common Dominant Reference Vectors

Dimensionality reduction is a standard technique for avoiding the severe problems the arise from the so-called "curse of dimensionality". Its usefulness is explained

by a wide spectrum of advantages, including the reduced number of features per sample and the improved space and time complexity of the developed algorithms. Consequently, its utilization in numerous machine learning tasks is beneficial and this also applies in the case of short text clustering.

The majority of the dimensionality reduction algorithms achieve their objective by introducing a mechanism for effectively projecting the original feature vectors onto a lower dimensional space. In this paper, we adopt this strategy with the aim of performing an initial clustering of the input samples. The main idea in VEPH is to project the input feature vectors of all similar documents onto the same space. Then, intuitively, those samples that fall into the same vector space, will be placed in the same cluster. According to the terminology of Section 3, the goal is to identify the samples having the same dominant reference vectors (RDVs) and, subsequently, group them in the same cluster.

This logic is decomposed into a series of steps. At first, we introduce a general hyper parameter K. In the sequel, we process each input $\mathbf{x}_i$ by forming all the possible projection vectors having $1, 2, \ldots, K$ components drawn from $\mathbf{x}_i$. In other words, we construct a set of projection vectors by picking combinations of $1, 2, \ldots, K$ components from the original vector $\mathbf{x}_i$.

Example 2. Consider an input vector $\mathbf{x} = (x_1, x_2, x_3, x_4)$ and let K be equal to 3. Then, the following projection vectors will be constructed: (x_1), (x_2), (x_3), (x_4), (x_1, x_2), (x_1, x_3), (x_1, x_4), (x_2, x_3), (x_2, x_4), (x_3, x_4), (x_1, x_2, x_3), (x_1, x_2, x_4), (x_1, x_3, x_4), and (x_2, x_3, x_4).

Now, from this set of candidate projection vectors, our intention is to identify $\mathbf{x}_i^*$; that is, the one which most accurately represents the initial feature vector $\mathbf{x}_i$. We call this vector as the *Dominant Projection Vector (DPV)* and its dimensional space will determine the cluster that will accommodate the respective document. The computation of the RDV of a document d_i will take place by assigning scores to all candidate projection vectors, and then, by selecting the one which achieved the highest score among the others.

Thus, we introduce the following scoring function to assign a score to the jth projection vector of an input sample $\mathbf{x_i}$:

$$S_{\mathbf{x}'_{ij}} = \frac{1}{l_{\mathbf{x}'_{ij}}} \log f_{\mathbf{x}'_{ij}} \sum_{\forall x_{ij} \in \mathbf{x}'_{ij}} x_{ij}^K. \tag{3}$$

In this equation, $f_{\mathbf{x}'_{ij}}$ is the number of the occurrences of $\mathbf{x}'_{ij}$; namely, how many times it has been constructed for all input vectors. As stated earlier, $l_{\mathbf{x}'_{ij}}$ is the length of $\mathbf{x}_{ij}$.

After all DPVs have been computed, the extraction of the respective DRVs is very easy. Recall that the DRV of d_i is a unit vector that lies in the same feature space as the DPV. In other words, it consists of the same components and its length is equal to 1. The procedure ends by grouping all the elements that share a common

DPV into the same cluster. This is the first stage of VEPH and leads to an initial formation of clusters.

Eq. 3 has been developed to improve the homogeneity and completeness of the produced clusters. The following discussion justifies this claim.

At first, it is obvious that long DPVs/DRVs lead to homogeneous clusters, because, in this case, the documents of the corresponding cluster will have more features (words) in common. Consequently, the summation term of Eq. 3 will assign higher scores to the longer projection vectors. In contrast, the native sparseness of short texts limits the probability that two documents have many words in common. This means that long DPVs/DRVs lead to clusters that tend to be incomplete. Thus, to limit this effect, we place the length of the projection vector in the denominator of Eq. 3. The number of the occurrences $f_{\mathbf{x}'_{ij}}$ of the projection vector has also been incorporated to enhance the completeness of the clusters: highly frequent DPVs are shared by numerous documents.

Consequently, long (short) Dominant Projection Vectors lead to clusters that tend to be homogeneous (inhomogeneous), but less (more) complete. Therefore, the projection vector that will be dominant must have the ideal length, so that it improves the clustering quality in terms of both homogeneity and completeness.

For the same reasons, the value of the hyper parameter K is also important. Recall that K controls the maximum number of features that will be present in the projection vectors; therefore, it directly affects their maximum lengths. Similarly to our previous discussion, large values of K will in general lead to homogeneous but incomplete clusters, and vice versa. A common setting for this hyper parameter is an integer value between 2 and 6; greater values have a negative impact in both clustering quality and execution time.

Now, if we combine the *tp-idf* vectorization method (Eq. 2) with the scores of Eq. 3, we obtain the final equation for assigning scores to the projection vectors:

$$S_{\mathbf{x}'_{ij}} = \frac{1}{l_{\mathbf{x}'_{ij}}} \log f_{\mathbf{x}'_{ij}} \sum_{\forall x_{ij} \in \mathbf{x}'_{ij}} \left(\log \frac{n}{f_j} \sum_{\forall t_j \in d_i} \frac{1}{\alpha + \log(p_{ij}+1)} \right)^K . \tag{4}$$

Example 3. Consider the documents of Table 2 and their respective vector representations (shown in the second column). Recall that according to our established notation, x_{ij} denotes the weight of the jth term of the ith document. For a setting $K = 2$, the sets of all possible vector projections are recorded in the third column. Now, let us assume that each projection vector is assigned a score based on Eq. 3, and that the dominant projection vectors of the forth column are computed.

Table 3 shows the Dominant Reference Vectors (DRVs) for each document, as they derive from the associated Dominant Projection Vectors. As mentioned earlier, the dimensionality of DRVs determines the cluster that will accommodate each document. Hence, the cluster of each document is reported in the last column of Table 3.

Table 2. An example of documents, their vector representations and their projection vectors.

Document	$\mathbf{x}$	$\mathbf{x}'$	DPV $\mathbf{x}^*$
$d_1 = \{t_1, t_2, t_3\}$	$\mathbf{x}_1 = (x_{11}, x_{12}, x_{13})$	(x_{11}), (x_{12}), (x_{13}) (x_{11}, x_{12}), (x_{11}, x_{13}),(x_{12}, x_{13})	$x_{11}\hat{\mathbf{t}}_1 + x_{12}\hat{\mathbf{t}}_2$
$d_2 = \{t_1, t_2, t_4\}$	$\mathbf{x}_2 = (x_{21}, x_{22}, x_{24})$	(x_{21}), (x_{22}), (x_{23}) (x_{21}, x_{22}), (x_{21}, x_{23}),(x_{22}, x_{23})	$x_{21}\hat{\mathbf{t}}_1 + x_{22}\hat{\mathbf{t}}_2$
$d_3 = \{t_1, t_2\}$	$\mathbf{x}_3 = (x_{31}, x_{32})$	(x_{31}), (x_{32}), (x_{31}, x_{32})	$x_{31}\hat{\mathbf{t}}_1 + x_{32}\hat{\mathbf{t}}_2$
$d_4 = \{t_3, t_4\}$	$\mathbf{x}_4 = (x_{41}, x_{42})$	(x_{41}), (x_{42}), (x_{41}, x_{42})	$x_{41}\hat{\mathbf{t}}_3 + x_{42}\hat{\mathbf{t}}_4$
$d_5 = \{t_3, t_4, t_5\}$	$\mathbf{x}_5 = (x_{51}, x_{52}, x_{53})$	(x_{51}), (x_{52}), (x_{53}) (x_{51}, x_{52}), (x_{51}, x_{53}),(x_{52}, x_{53})	$x_{51}\hat{\mathbf{t}}_3 + x_{52}\hat{\mathbf{t}}_4$

Table 3. Dominant Reference Vectors and clustering.

Document	DPV $\mathbf{x}^*$	DRV $\hat{\mathbf{x}}^*$	**Cluster**
$d_1 = \{t_1, t_2, t_3\}$	$x_{11}\hat{\mathbf{t}}_1 + x_{12}\hat{\mathbf{t}}_2$	$\hat{\mathbf{t}}_1 + \hat{\mathbf{t}}_2$	c_1
$d_2 = \{t_1, t_2, t_4\}$	$x_{21}\hat{\mathbf{t}}_1 + x_{22}\hat{\mathbf{t}}_2$	$\hat{\mathbf{t}}_1 + \hat{\mathbf{t}}_2$	c_1
$d_3 = \{t_1, t_2\}$	$x_{31}\hat{\mathbf{t}}_1 + x_{32}\hat{\mathbf{t}}_2$	$\hat{\mathbf{t}}_1 + \hat{\mathbf{t}}_2$	c_1
$d_4 = \{t_3, t_4\}$	$x_{41}\hat{\mathbf{t}}_3 + x_{42}\hat{\mathbf{t}}_4$	$\hat{\mathbf{t}}_3 + \hat{\mathbf{t}}_4$	c_2
$d_5 = \{t_3, t_4, t_5\}$	$x_{41}\hat{\mathbf{t}}_3 + x_{42}\hat{\mathbf{t}}_4$	$\hat{\mathbf{t}}_3 + \hat{\mathbf{t}}_4$	c_2

5. Cluster Refinement

After the clustering of the previous stage is completed, the refinement process follows. This procedure consists of two successive steps, namely: i) a split phase, where the most dissimilar elements are removed from their respective clusters; and ii) a merge phase that unifies entire clusters comprising similar elements. In the following subsections we present each of these phases, and we emphasize on the points that discriminate them from the respective phases of VEPHC.[4]

5.1. *Split Phase*

The goal of this phase is to improve the homogeneity of the clusters that were formed after the computation of the DPVs and the subsequent grouping of the input documents according to their common DRVs.

There are two prerequisites for this procedure: The first one requires a solid definition of the similarity between an input text vector $\mathbf{x}$ and a cluster c. Various methods have been proposed for this purpose such as simple and complete record linkage, Ward, computation of the Euclidean distance between $\mathbf{x}$ and the centroid of c, etc. Here, we adopt the same strategy as in VEPHC. Namely, we define the similarity between an input vector $\mathbf{x}$ and a cluster c as the cosine similarity between $\mathbf{x}$ and the clustroid $\mathbf{u}_c$. Recall that the clustroid of c is defined as the item (vector) with the greatest similarity with the rest of the elements of c.

The second requirement is the introduction of a second hyper-parameter, which we name *split threshold* T_s. Notice that T_s is a parameter that substantially affects the homogeneity of the clusters, because it determines which (dissimilar) elements must be evicted from the clusters. Moreover, T_s is a similarity threshold and, as such, it may receive any real value in the range $(0, 1)$.

Algorithm 1: The split phase of the refinement stage of VEPH

```
1  for each cluster c ∈ C do
2  |  u_c ← compute the clustroid of c by using Algorithm 3;
3  end
4  for each cluster c ∈ C do
5  |  for each document vector x ∈ c do
6  |  |  sim_{x,u_c} ← cos(x, u_c);
7  |  |  if sim_{x,u_c} < T_s then
8  |  |  |  c ← c − {x};
9  |  |  |  create cluster c_new ← ∅;
10 |  |  |  c_new ← c_new ∪ {x};
11 |  |  |  u_{c_new} ← x;
12 |  |  |  C ← C ∪ {c_new};
13 |  |  end
14 |  end
15 |  u_c ← (re)compute the clustroid of c by using Algorithm 3;
16 end
```

After both of these requirements have been satisfied, the split phase iterates through all clusters of C. For each cluster $c \in C$, it removes all the elements whose similarity with $\mathbf{u}_c$ is smaller than T_s. In VEPHC, the split/transfer algorithm immediately searches for other, more similar clusters that may accommodate the evicted elements. However, here we propose a different approach. More specifically, for each evicted element, a new cluster is created and the evicted element is inserted into this new cluster. It is clarified that all the new clusters which are created during this phase are accommodating only one item, that is, they are *singleton clusters*.

Algorithm 1 contains the most important steps of the split process. Initially, the clustroids of all clusters of C are computed (lines 1–3), by using Algorithm 3 that is presented in the sequel. The second loop in lines 4–16 iterates again through all clusters of C, and for each cluster c, it calculates the cosine similarity of all of its items $\mathbf{x}$ with the clustroid $\mathbf{u}_c$ (line 6). If the similarity between $\mathbf{x}$ and $\mathbf{u}_c$ is smaller than the value of the split threshold T_s, then the following operations take place:

(1) The dissimilar element $\mathbf{x}$ is removed from its cluster c (line 8);
(2) A new cluster c_{new} is created (line 9);
(3) $\mathbf{x}$ is inserted to c_{new} (lines 10–11), and becomes its new clustroid;
(4) c_{new} becomes a member of the universe C (line 12); and
(5) The clustroid of c is recomputed (line 15).

Consequently, at the end of the split phase, the universe C is enriched with new singleton clusters, whereas the old ones will now be more homogeneous, due to the eviction of their most dissimilar elements.

Algorithm 2: The merge phase of the refinement stage of VEPH

```
1  msc_mat[*] ← NULL, sim_mat[*] ← −1;
2  for each cluster c ∈ C do
3  |  msc_mat[c] ← compute the most similar cluster (MSC);
4  |  sim_mat[c] ← similarity with MSC;
5  end
6  while TRUE do
7  |  cand_1 ←NULL, cand_2 ←NULL;
8  |  for each cluster c ∈ C do
9  |  |  sim_max ← 0;
10 |  |  if sim_mat[c] > sim_max then
11 |  |  |  sim_max ← sim_mat[c];
12 |  |  |  cand_1 ← c;
13 |  |  |  cand_2 ← msc_mat[c];
14 |  |  end
15 |  end
16 |  if sim_max > T_m then
17 |  |  cand_1 ← cand_1 ∪ cand_2;
18 |  |  u_cand_1 ← (re)compute the clustroid of cand_1;
19 |  |  C ← C − {cand_2};
20 |  |  msc_mat[cand_1] ← (re)compute most similar cluster (MSC);
21 |  |  sim_mat[cand_1] ← similarity with MSC;
22 |  |  msc_mat[cand_2] ← NULL;
23 |  |  sim_mat[cand_2] ← −1;
24 |  else
25 |  |  break;
26 |  end
27 end
```

5.2. *Merge Phase*

In contrast to the previous phase, the cluster merge phase aims at the enhancement of the completeness of the clusters of C, including the recently created ones.

The merge phase is executed directly upon the completion of the split phase. The merging operation takes place in an iterative hierarchical fashion, and, particularly, its agglomerative version. The entire process is controlled by a *merge threshold* $T_m \in (0, 1)$ that determines the termination of the iterations. This procedure is described in details by Algorithm 2.

At first, two column $|C| \times 1$ matrices are required: the first one, *msc_mat*, stores pointers to the most similar clusters of each cluster $c \in C$, whereas the second one, *sim_mat*, stores these maximum similarity values. In accordance to other works,

the similarity between two clusters can be represented quite effectively by the cosine similarity between their clustroid elements. Consequently, the ith element of *sim_mat* contains the cosine similarity between the clustroid of c_i and the clustroid of its most similar cluster.

After the initialization of the two aforementioned matrices (lines 1–5), the iterative cluster merging begins (lines 6–27). In steps 8–15 the algorithm detects the pair of the most similar clusters in the universe C. Let $cand_1$ and $cand_2$ be these two clusters. If their similarity exceeds the value of the merge threshold T_m, then the following operations are performed (lines 16–24):

(1) $cand_1$ is merged with $cand_2$, the result is stored in $cand_1$ and its clustroid is recomputed (lines 17–18);
(2) $cand_2$ is removed from C (line 19); and
(3) The two auxiliary matrices *msc_mat* and *sim_mat* are updated accordingly (lines 20–23).

On the other hand, in case the similarity between $cand_1$ and $cand_2$ does not exceed the value of the merge threshold T_m, then we conclude that no other cluster merging may occur. This triggers the process termination condition, and the execution flow exits the loop.

As mentioned earlier, the goal of the refinement stage of VEPH is the improvement of the quality of clustering that was achieved by the initial vector projection stage. Therefore, the split phase is a cleansing process that enhances homogeneity, because it evicts the most dissimilar items from the clusters. On the other hand, the merge phase is a procedure designed to improve completeness, since it brings together the most similar elements. Notice that the effectiveness of both phases is affected by the values of the two aforementioned similarity thresholds T_s and T_m.

5.3. *Complexity Analysis*

In this subsection we present a time complexity analysis for the worst-case scenario of the refinement stage of VEPH.

Algorithm 1 represents the split phase of VEPH and comprises two loops: The first one computes the clustroids of the clusters of C using Algorithm 3 and its worst-case complexity is $O(|C|n_c^2)$, where n_c denotes the maximum number of items included in a cluster. The second loop removes the most dissimilar elements from all clusters and its worst-case complexity is $O(|C|n_c)$. Therefore, the total time complexity of the worst-case scenario of Algorithm 1 is computed as $O(|C|n_c^2 + |C|n_c) = O(|C|n_c^2)$. Apparently, the dominant part in the split phase is the (re)computation of the clustroid elements.

Regarding Algorithm 2, the time complexity in the worst-case scenario is upper bounded by $O(|C|n_c^2 + |C|^2)$. More specifically, the initialization of the two matrices *msc_mat* and *sim_mat* has a cost of $O(|C|)$ (line 1) plus $O(|C|^2)$ (lines 2–5). In addition, the hierarchical merging process has a worst-case cost of $O(|C|(|C| + n_c^2))$.

Algorithm 3: Clustroid computation for a cluster c

```
sim_max ← 0;
for each element x ∈ c do
    sim ← 0;
    for each element x' ∈ c do
        sim ← sim + cos(x, x');
    end
    if sim > sim_max then
        sim_max ← sim;
        u_c ← x;
    end
end
return u_c;
```

6. Experimental Evaluation

In this section, we analyze the results of the experimental evaluation of the proposed method. The presentation is divided into four subsections that describe the employed datasets, the utilized evaluation measures, the adversary clustering algorithms, and the effectiveness measurements themselves, respectively. There is also a special subsection that contains a detailed study on the sensitivity of VEPH against fluctuations of the values of its hyper parameters.

6.1. *Datasets*

We begin with the 3 datasets that were employed to evaluate the usefulness of the proposed method. At first, News Aggregator[b] (NA) is a well-established dataset that contains about 423 thousand article titles. Although it is not quite large (it occupies approximately 100 MB in uncompressed form), several adversary clustering methods (such as the Agglomerative clustering algorithm and DBSCAN) failed to complete their task within a reasonable time. For this reason, we randomly selected 50,138 news records grouped into 823 stories (namely, clusters) for our tests.

The second dataset is called TitleSet (TSET) and has been used in several state-of-the-art works in the literature.[2,17] It was extracted from a collection of news titles, from a snapshot of Google News on November 27th, 2013. It consists of 11,108 article titles grouped in 152 news stories.

The third dataset was manually constructed by crawling 9 product categories from PriceRunner (PRR[c]), an online service for comparing product offers. We emphasize that in this case we are not interested in grouping these products into one of

[b]https://www.kaggle.com/uciml/news-aggregator-dataset
[c]https://www.kaggle.com/lakritidis/product-classification-and-categorization

Table 4. Experimental datasets.

Dataset	**Abbreviation**	n	$\|C\|$	$l_{\max}$	l_{ave}
News Aggregator	NA	50138	823	15	7.05
TitleSet	TSET	11108	152	14	6.10
PriceRunner	PRR	35311	13233	41	8.23

these nine categories.[33] Instead, the goal is to construct clusters of identical products, similarly to the operation of an entity matching algorithm. In this way, the platform facilitates product comparison. The dataset contains 35311 titles that correspond to 13233 distinct products, and has been utilized by multiple works in the relevant literature.[4,11,12]

Table 4 summarizes the characteristics of the three datasets. In the last two columns we report the maximum ($l_{\max}$) and the average (l_{ave}) lengths (in numbers of words) of the input short texts, respectively.

6.2. *Evaluation Measures*

There is a multitudinous family of measures for the evaluation of the performance of the clustering algorithms. In this article, VEPH and its adversary methods are evaluated by employing the two most popular among them: the $F1$ score and Normalized Mutual Information (NMI).

The first one is a well-known metric that combines Precision and Recall and has been utilized in many applications beyond clustering (e.g. evaluation of IR systems). It is defined as follows:

$$F1 = \frac{2PR}{P+R}, \tag{5}$$

where P and R represent the Precision and the Recall of the algorithm. To calculate them, a procedure that iterates through all ground truth clusters C_{GT} was performed. For each element $\mathbf{x}$ within a ground truth cluster $c \in C_{GT}$, we recorded one pairwise match record $(\mathbf{x}, \mathbf{x}_r)$ with each of the rest of the elements $\mathbf{x}_r$ of c. At the end of the iteration, a database including a set of such pairwise match records was created. In the sequel, we repeated this process for the cluster universe C that was produced by each attested method. Eventually, we compared the ground truth pairwise matches against the ones of each method, and we measured the number of the True/False Positive/Negative records.

Regarding the second evaluation measure, NMI is essentially a method for estimating the amount of mutual information shared between the cluster assignments in C and C_{GT}:

$$NMI(C_{GT}, C) = \frac{2I(C_{GT}; C)}{H(C_{GT}) + H(C)}, \tag{6}$$

where $H(C_{GT})$ and $H(C)$ denote the entropies of the ground truth and clustered sets, respectively. Additionally, $I(C_{GT}; C) = H(C) - H(C|C_{GT})$ represents the amount of mutual information shared between the ground truth and clustered sets

Table 5. The attested clustering algorithms accompanied by the experimental setup of their hyper parameters.

Algorithm	Setup
VEPH	$K = 2$, $K = 6$
VEPHC	$K = 2$, $K = 6$
GSDMM-1	$\alpha = \beta = 0.1$, $k = 3\|C\|/2$
GSDMM-2	$\alpha = 0.001$, $\beta = 0.01$, $k = 3\|C\|/2$
Spherical k-Means	$k = \|C\|$, $I = 10$
vk-Means	$k = \|C\|$, $I = 10$
k-Means	$k = \|C\|$, $I = 10$
Hierarchical (Agglomerative, HAC)	–
Leader Clustering	–
DBSCAN	$minPts = 2$

C_{GT} and C. A value of NMI that is close to 0 reveals no, or little mutual information between two compared sets; in contrast, a value of 1 indicates that the two sets are perfectly correlated.

6.3. *Clustering Algorithms and Setup*

The performance of VEPH was compared against 4 short text and 4 generic data clustering algorithms in terms of both clustering quality and execution time. Table 5 enlists these 8 methods, accompanied by the experimental setup of their respective hyper parameters. Notice that the values displayed in the second column consistently led the associated algorithms to perform optimally in terms of clustering quality ($F1$ scores). The following list briefly describes the 8 adversary algorithms:

- VEPHC[4] is the predecessor algorithm of VEPH, with a simple tf-idf input vectorization method and a different refinement stage (as described in Section 5).
- GSDMM[23] is a collapsed Gibbs Sampling algorithm for the Dirichlet Multinomial Mixture model. Here we report performance measurements by setting the hyper parameters i) as it was proposed in the original paper (GSDMM-1), and ii) by adopting the values utilized in a public implementation[d] (GSDMM-2).
- Spherical k-Means[1,2] is an adaptation of the traditional k-Means algorithm for text datasets. In the context of this experimental evaluation, all k-Means variants were executed by setting the initial number of clusters equal to the actual number of clusters ($k = |C|$, a rather unrealistic scenario in real-world unsupervised tasks), and the maximum number of iterations equal to 10.
- vk-Means constitutes our own variant of k-Means for text datasets. In vk-Means the centroid vectors and the Euclidean distances of the original algorithm have been replaced by the clustroid elements and the cosine similarities, respectively.
- k-Means is a traditional space partitioning algorithm that iteratively groups the involved data points, based on their Euclidean distance from the (moving) centroids of the clusters.

[d]https://github.com/rwalk/gsdmm

- Hierarchical Clustering is one of the oldest and most successful (albeit slow) clustering techniques. It has two equivalent versions, the divisive, and the agglomerative one. In this work we implemented the latter version (HAC), that requires the existence of a similarity threshold to determine whether two clusters are similar enough to be merged. We executed the agglomerative (and the next two) methods 10 times with different values of this threshold (in the range $[0, 1]$) and in the following results, we report the performance of the best run.
- Leader Clustering[34] is a method based on a special point in the cluster, called the Leader element. This method is also subject to the aforementioned similarity threshold, with the aim of quantifying the relationship of a data point with the Leaders of the clusters.
- DBSCAN[35] is a well-established density-based clustering algorithm. The $minPts$ hyper parameter that determines the minimum number of elements per cluster was set equal to 2. This method is also subject to the aforementioned similarity threshold.

All algorithms were implemented in C++ and compiled by gcc 7.5.0 with O3 optimization. Public versions of the code were integrated within the SHTECLib library[e]. The code was executed on a machine equipped with an Intel CoreI7-7700 and 32GB of RAM, running Linux Mint 19.03.

6.4. *Hyper Parameters Study*

As mentioned earlier, VEPH includes four hyper parameters. The first one is α, a real value in the range $[0, 1]$ that regulates the contribution of the position of each word in the vector coefficients of the input documents. Typical settings for α that consistently led to a satisfactory performance of VEPH are $0.3 \leq \alpha \leq 0.5$. In the experiments that follow we set $\alpha = 0.3$.

The next hyper parameter to examine is K, of which the (integer) value determines the maximum length of the projection vectors during the first stage of VEPH. Figure 1 illustrates the performance fluctuations in terms of Precision, Recall, $F1$ score and NMI, as K is modified from 2 to 6. Notice that in order to accurately study the actual effects of K, we only report the performance of the first stage of VEPH, that is, without applying the refinement stage of Section 5. Consequently, the three diagrams of Fig. 1 represent the effectiveness of the first stage of VEPH when it is applied in an isolated manner on each of the three test datasets.

Regarding the first two datasets (NA and TSET), the form of the curves is quite similar. Namely, in both cases, VEPH achieves its best performance for the smallest possible value of K (i.e., $K = 2$) in terms of both $F1$ and NMI. As the value of K gradually increases, these two measures are beginning to fall. Eventually, for $K = 6$, $F1$ becomes almost zero in both datasets. On the other hand, the degradation of NMI is much smoother. Hence, for $K = 2$, the values of NMI for NA and TSET

[e]https://github.com/lakritidis/SHTECLib

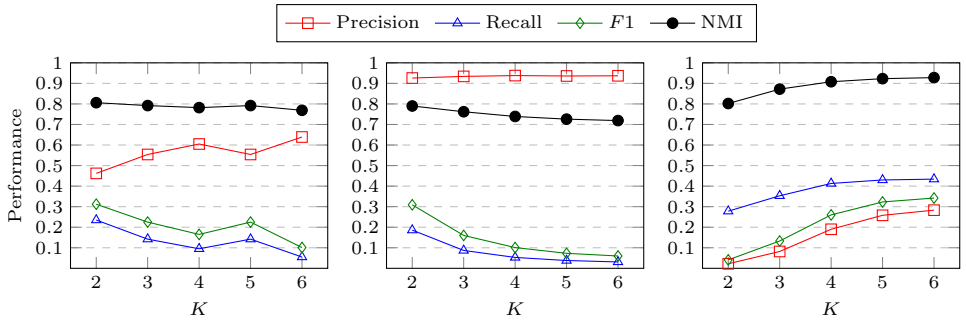

Fig. 1. Graphical representation of the clustering performance of VEPH vs. the hyper parameter K. The three figures concern the i) News Aggregator (left), ii) TitleSet (center), and iii) Pricerunner (right) datasets.

are 0.806 and 0.790 respectively, whereas for $K = 6$, these values become 0.769 and 0.719 respectively.

The situation is reversed in the case of PRR, where the worst and best performances are observed for $K = 2$ and $K = 6$ respectively. More specifically, for $K = 2$ the $F1$ score is almost zero, and rises to approximately 0.34 for $K = 6$. Similar improvements are also valid for NMI, Precision, and Recall.

Now, let us focus on the refinement stage of VEPH and the two similarity thresholds T_s (split) and T_m (merge threshold). Recall that a document will leave a cluster if its similarity with the clustroid is smaller than T_s. Also, two clusters having a similarity value that is greater than T_m will finally be merged to form a single cluster. Figure 2 depicts three heat maps (one for each dataset) of the $F1$ score of VEPH, when both T_s and T_m are modified in the range $[0.1, 0.9]$.

Similarly to the case of K, the observations on the first two datasets are similar. VEPH achieves its best performance for small values of T_s (about 0.1–0.2) and large values of T_m (roughly 0.7–0.8). On the other hand, in PRR the clustering effectiveness is maximized for moderate values of T_s (namely, $\simeq$ 0.5–0.6) and small values of T_m ($\simeq$ 0.1). This means that it is difficult for a product title to leave its initial cluster; if it does, its integration with another existing clustering has a moderate probability.

In general, product clustering is a significantly more challenging problem than the simple short text clustering.[11,12] The reason is that very similar product titles often refer to different products, and vice versa. This explains the considerable differences between the optimal hyper parameter values in PRR, and the other two datasets. In product clustering, the projection vectors must have more components (i.e., K must have a greater value) to ensure that only highly similar documents are inserted into the same cluster. For this reason, the product titles should not be able to leave a cluster as easily as in the case of standard short text documents. This explains why the optimal T_m was significantly smaller in the case of PRR.

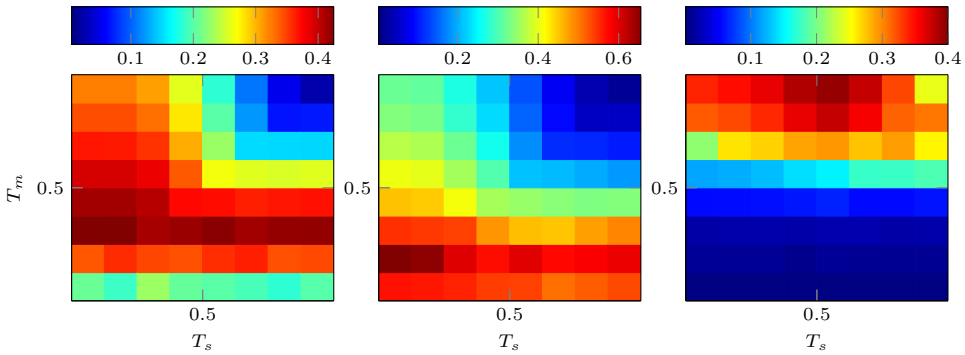

Fig. 2. Heat map of the $F1$ scores of VEPH against variations of the two similarity thresholds T_s and T_m. The three figures concern the i) News Aggregator (left), ii) TitleSet (center), and iii) Pricerunner (right) datasets.

6.5. *Performance Evaluation*

We proceed with the presentation of the performance evaluation of VEPH. We examined both the effectiveness (i.e., clustering quality) and the efficiency (running times) of the proposed method. Table 6 contains measurements on the three aforementioned datasets, compared with the adversary clustering methods that were described above. These measurements concern the achieved $F1$ and NMI scores and the execution times of all the involved algorithms.

VEPH achieved the best clustering performance among all examined methods in two of the three test datasets, namely, NA and PRR. In both cases, the strongest opponent was its predecessor algorithm. More specifically, in terms of $F1$ scores, VEPH outperformed VEPHC by small to moderate margins of 1.2% (in NA) and 5.2% (in PRR). Regarding NMI, both methods were equally effective. According to our previous study on hyper parameters, the optimal performance of VEPH in NA was obtained with a setup of $K = 2$, $T_s = 0.2$, and $T_m = 0.3$. Regarding PRR, these values were $K = 6$, $T_s = 0.5$, and $T_m = 0.9$.

Surprisingly, the second strongest method in both datasets was Agglomerative clustering (HAC), achieving $F1$ scores equal to 0.454 in NA, and 0.314 in PRR. In other words, VEPH outperformed HAC by about 5% and 29% respectively. Notice that the differences in the values of NMI were slightly smaller.

Regarding GSDMM, the second configuration scored $F1 = 0.424$ and $NMI = 0.82$ on the NA dataset. These values are smaller by 13% and 4% than the respective ones of VEPH. On the other hand, GSDMM with a setting of $\alpha = \beta = 0.1$ (i.e. GSDMM-1) was rather ineffective, scoring $F1 = 0.171$ and $NMI = 0.75$. Nevertheless, both configurations of GSDMM had a disappointing behavior on the PRR dataset and failed to produce a decent clustering of the input records. Indicatively, the measured $F1$ scores were close to zero, whereas the NMI of GSDMM-1 was the worst among all the examined methods.

However, GSDMM was the highest performing method on the TitleSet dataset.

Table 6. Performance evaluation of the clustering methods of Table 5 on the datasets of Table 4.

Algorithm	News Aggregator			TitleSet			Pricerunner		
	$F1$	NMI	Time	$F1$	NMI	Time	$F1$	NMI	Time
VEPH	0.477	0.853	9.4	0.685	0.865	1.2	0.405	0.946	326.0
VEPHC	0.471	0.851	10.2	0.574	0.830	1.0	0.385	0.946	305.1
GSDMM-1	0.171	0.753	390.4	0.719	0.872	13.3	0.002	0.404	3246.2
GSDMM-2	0.424	0.820	453.9	0.495	0.829	14.0	0.008	0.631	3317.3
Sph. k-Means	0.335	0.757	119.6	0.603	0.832	4.0	0.226	0.903	426.1
vk-Means	0.154	0.614	104.7	0.375	0.721	1.8	0.220	0.900	484.4
k-Means	0.028	0.697	104.3	0.072	0.634	3.9	0.048	0.399	494.2
Agglomerative	0.454	0.835	2492.8	0.597	0.836	76.2	0.314	0.935	166.7
Leader	0.284	0.778	17.1	0.400	0.745	0.2	0.306	0.934	17.2
DBSCAN	0.256	0.790	130.5	0.294	0.603	5.8	0.055	0.425	58.4

Its $F1$ and NMI scores were 0.719 and 0.872, respectively. VEPH was in the second place this time, but it was competitive enough, since it was outperformed by a margin no greater than 5%. In addition, in this dataset we measured the greatest difference between the performances of VEPH and VEPHC. More specifically, the former outperformed the latter by 19% and 4% in terms of $F1$ and NMI, respectively. VEPHC was also outperformed by HAC and Spherical k-Means by 4% and 5% respectively.

To summarize, VEPH yielded satisfactory clustering quality in all cases. It outperformed all its adversary algorithms on two of the three datasets (NA and PRR), whereas on the third one, it was marginally outperformed by GSDMM only. On the other hand, GSDMM had the worst performance among the 9 attested methods on the PRR dataset (near zero $F1$ score in both configurations) and it was also worse than HAC on the News Aggregator dataset. Regarding the other algorithms that were included in the experiments, HAC, Spherical k-Means and Leader Clustering performed decently on all datasets, however, they were all outperformed by VEPH.

Finally, we study VEPH from the efficiency perspective. Before we discuss the respective columns of Table 6, it is required that we clarify some points. At first, the presented durations concern the basic C++ implementations of SHTECLib. Therefore, we compare software solutions that were developed by using the same programming language. Second, we have performed some optimizations in all cases (e.g. string comparisons, temporary auxiliary matrices for HAC, DBSCAN and k-Means, etc.), but we did not implement several exhaustive optimizations that have been proposed in the relevant literature.

The third point is that the running times of almost all algorithms are significantly affected by the values of their hyper parameters. For instance, the value of the hyper parameter K has a great impact on the execution time of VEPH, because the larger it is, the more projection vectors are generated. In addition, the values of the thresholds T_s and T_m affect the duration of the refinement stage, since they essentially determine the number of elements that leave the clusters and the number of clusters to be merged.

After these comments, we observe that in the cases of NA and TSET (where we

set $K = 2$), VEPH and VEPHC were either the fastest methods (e.g. NA), or the second fastest (TSET). In particular, VEPH completed its tasks in about 9.4 and 1.2 seconds, respectively. On the other hand, on PRR (where K was set equal to 6), VEPH consumed about 5.5 minutes and was outperformed by Leader Clustering ($17''$), DBSCAN ($58''$), HAC ($166.7''$) and VEPHC ($305''$). This verifies the great impact of K in the overall running time of VEPH.

Leader clustering was the fastest method in two out of the three tests (it was outperformed by VEPH on the News Aggregator dataset). In contrast, Agglomerative clustering (on the NA & TSET datasets) and GSDMM (on PRR) were the slowest ones.

7. Conclusions

In this paper we introduced VEPH, an unsupervised algorithm for clustering short text documents. Initially, we verified the weaknesses of the traditional tf-idf text vectorization technique when it is applied on short documents. We proposed a novel text vectorization method that drops the tf term and replaces it by the position of a word in a document.

The first stage of VEPH projects the initial text vectors onto a lower dimensional space. It does so by initially setting an integer hyper parameter K. Then, for each input vector $\mathbf{x}$, it forms and scores all the $1, 2, \ldots K$-dimensional projection vectors of $\mathbf{x}$. The projection vector $\mathbf{x}^*$ that was assigned the highest score (called Dominant Projection Vector, DPV) is subsequently utilized to construct the Dominant Reference Vector (DRV) from it. In short, the DRV is a unit vector with the same components as the DPV. Finally, all documents having common DPVs are grouped within the same cluster.

In the sequel, a second split/merge refinement stage is performed. During this stage, the most dissimilar elements are evicted from their respective clusters to form new, singleton clusters. In the second phase of the refinement stage, the old and the singleton clusters are merged together by adopting a hierarchical logic that is similar to the Agglomerative clustering algorithm.

VEPH was extensively attested against 8 state-of-the-art clustering methods by employing three standard, real-world datasets. The experimental evaluation derived the following conclusions:

- VEPH performed satisfactorily on the three attested datasets. In particular, it outperformed all the examined clustering algorithms with the exception of GSDMM on the second dataset.
- In comparison to GSDMM, VEPH had 2 wins and one loss. Nevertheless, the experiments have shown that VEPH produced qualitative clusterings on all the examined datasets. In contrast, GSDMM exhibited a surprisingly bad behavior on the Pricerunner dataset, achieving $F1$ scores that were close to zero. Furthermore, VEPH was much faster in all tests.
- Agglomerative clustering also produced a satisfactory clustering for all cases.

However, it was less effective and less efficient than VEPH. Regarding the other adversary methods, the two most notable were Spherical k-Means and Leader Clustering that achieved decent performance in all cases. The latter was the fastest method in two of the three tests.

References

1. I. S. Dhillon and D. S. Modha, Concept decompositions for large sparse text data using clustering, *Machine Learning* **42**(1-2) (2001) 143–175.
2. C. Jia, M. B. Carson, X. Wang and J. Yu, Concept decompositions for short text clustering by identifying word communities, *Pattern Recognition* **76** (2018) 691–703.
3. C. T. Zheng, C. Liu and H. San Wong, Corpus-based topic diffusion for short text clustering, *Neurocomputing* **275** (2018) 2444–2458.
4. L. Akritidis, M. Alamaniotis, A. Fevgas and P. Bozanis, Confronting sparseness and high dimensionality in short text clustering via feature vector projections, in *Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence* (IEEE, 2020), pp. 813–820.
5. M. Sahami and T. D. Heilman, A Web-based kernel function for measuring the similarity of short text snippets, in *Proceedings of the 15th International Conference on World Wide Web* (ACM, 2006), pp. 377–386.
6. S. Banerjee, K. Ramanathan and A. Gupta, Clustering short texts using Wikipedia, in *Proceedings of the 30th ACM Conference on Research and Development in Information Retrieval* (ACM, 2007), pp. 787–788.
7. D. Pinto, P. Rosso and H. Jiménez-Salazar, A self-enriching methodology for clustering narrow domain short texts, *The Computer Journal* **54**(7) (2011) 1148–1165.
8. G. Rizos, K. Hemker and B. Schuller, Augment to prevent: short-text data augmentation in deep learning for hate-speech classification, in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (ACM, 2019), pp. 991–1000.
9. X. Yan, J. Guo, Y. Lan and X. Cheng, A biterm topic model for short texts, in *Proceedings of the 22nd International Conference on World Wide Web* (ACM, 2013), pp. 1445–1456.
10. S. Seifzadeh, A. K. Farahat, M. S. Kamel and F. Karray, Short-text clustering using statistical semantics, in *Proceedings of the 24th International Conference on World Wide Web* (ACM, 2015), pp. 805–810.
11. L. Akritidis and P. Bozanis, Effective unsupervised matching of product titles with k-combinations and permutations, in *Proceedings of the 14th IEEE International Conference on Innovations in Intelligent Systems and Applications* (IEEE, 2018), pp. 1–10.
12. L. Akritidis, A. Fevgas, P. Bozanis and C. Makris, A self-verifying clustering approach to unsupervised matching of product titles, *Artificial Intelligence Review* **53** (2020) 4777–4820.
13. J. Xu, B. Xu, P. Wang, S. Zheng, G. Tian and J. Zhao, Self-taught Convolutional Neural Networks for short text clustering, *Neural Networks* **88** (2017) 22–31.
14. W. Xu and Y. Gong, Document clustering by concept factorization, in *Proceedings of the 27th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Association for Computing Machinery, 2004), pp. 202–209.
15. M. Chen, Q. Wang and X. Li, Adaptive projected matrix factorization method for data clustering, *Neurocomputing* **306** (2018) 182–188.
16. D. D. Lee and H. S. Seung, Learning the parts of objects by non-negative matrix factorization, *Nature* **401**(6755) (1999) 788–791.

17. X. Yan, J. Guo, S. Liu, X. Cheng and Y. Wang, Learning topics in short texts by Non-Negative Matrix Factorization on term correlation matrix, in *Proceedings of the 2013 SIAM International Conference on Data Mining* (SIAM, 2013), pp. 749–757.
18. X. Yan, J. Guo, S. Liu, X.-q. Cheng and Y. Wang, Clustering short text using ncut-weighted non-negative matrix factorization, in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (ACM, 2012), pp. 2259–2262.
19. F. Shang, L. Jiao and F. Wang, Graph dual regularization non-negative matrix factorization for co-clustering, *Pattern Recognition* **45**(6) (2012) 2237–2250.
20. J. Ye and Z. Jin, Dual-graph regularized concept factorization for clustering, *Neurocomputing* **138** (2014) 120–130.
21. D. M. Blei, A. Y. Ng and M. I. Jordan, Latent dirichlet allocation, *Journal of Machine Learning Research* **3** (2003) 993–1022.
22. J. Kumar, J. Shao, S. Uddin and W. Ali, An online semantic-enhanced Dirichlet model for short text stream clustering, in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (Association for Computational Linguistics, July 2020), pp. 766–776.
23. J. Yin and J. Wang, A Dirichlet multinomial mixture model-based approach for short text clustering, in *Proceedings of the 20th ACM Conference on Knowledge Discovery and Data Mining* (ACM, 2014), pp. 233–242.
24. J. Xu, P. Wang, G. Tian, B. Xu, J. Zhao, F. Wang and H. Hao, Short text clustering via Convolutional Neural Networks, in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing* (Association for Computational Linguistics, 2015), pp. 62–69.
25. L. Bazzani, A. Bergamo, D. Anguelov and L. Torresani, Self-taught object localization with Deep Networks, in *Proceedings of the 2016 IEEE Winter Conference on Applications of Computer Vision* (IEEE, 2016), pp. 1–9.
26. C. Wei, S. Luo, X. Ma, H. Ren, J. Zhang and L. Pan, Locally embedding autoencoders: a semi-supervised manifold learning approach of document representation, *PloS one* **11**(1) (2016) p. e0146672.
27. A. Hadifar, L. Sterckx, T. Demeester and C. Develder, A self-training approach for short text clustering, in *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)* (Association for Computational Linguistics, 2019), pp. 194–199.
28. L. Wang, Y. Jia and W. Han, Instant message clustering based on extended Vector Space Model, in *Proceedings of the 2nd International Symposium on Intelligence Computation and Applications* (Springer, Berlin, Heidelberg, 2007), pp. 435–443.
29. X. Hu, N. Sun, C. Zhang and T.-S. Chua, Exploiting internal and external semantics for the clustering of short texts using world knowledge, in *Proceedings of the 18th ACM International Conference on Information and Knowledge Management* (ACM, 2009), pp. 919–928.
30. D. Milne, O. Medelyan and I. H. Witten, Mining domain-specific thesauri from Wikipedia: A case study, in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence* (IEEE/ACM, 2006), pp. 442–448.
31. A. Rangrej, S. Kulkarni and A. V. Tendulkar, Comparative study of clustering techniques for short text documents, in *Proceedings of the 20th International Conference Companion on World Wide Web* (ACM, 2011), pp. 111–112.
32. S. Likhitha, B. Harish and H. K. Kumar, A detailed survey on topic modeling for document and short text data, *International Journal of Computer Applications* **178**(39) (2019) 1–9.

33. L. Akritidis, A. Fevgas and P. Bozanis, Effective products categorization with importance scores and morphological analysis of the titles, in *Proceedings of the 30th IEEE International Conference on Tools with Artificial Intelligence* (IEEE, 2018), pp. 213–220.
34. H. Spath, *Cluster analysis algorithms for data reduction and classification of objects* (Ellis Horwood Chichester, 1980).
35. M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, A density-based algorithm for discovering clusters in large spatial databases with noise, in *Proceedings of the 2nd International Confernece on Knowledge Discovery and Data Mining* (AAAI Press, 1996), pp. 226–231.