Confronting Sparseness and High Dimensionality in Short Text Clustering via Feature Vector Projections

Leonidas Akritidis
School of Science and Technology
Int'l Hellenic University
Thessaloniki, Greece
Email: lakritidis@ihu.gr

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

Athanasios Fevgas
Department of Electrical and Computer Engineering
University of Thessaly
Volos, Greece
Email: fevgas@e-ce.uth.gr

Panayiotis Bozanis
School of Science and Technology
Int'l Hellenic University
Thessaloniki, Greece
Email: pbozanis@ihu.gr

***Abstract*—Short text clustering is a popular problem that focuses on the unsupervised grouping of similar short text documents, or entitled entities. Since the short texts are currently being utilized in a vast number of applications, the problem in question has been rendered increasingly significant in the past few years. The high cluster homogeneity and completeness are two among the most important goals of all data clustering algorithms. However, in the context of short texts, their fulfilment is particularly difficult, because this type of data is typically represented by sparse vectors that collectively comprise a very high dimensional space. In this article we introduce VEPHC, a two-stage clustering algorithm designed to confront the sparseness and high dimensionality traits of short texts. During the first stage (or else, the VEP part), the initial feature vectors are projected onto a lower dimensional space by constructing and scoring variable-sized combinations of features (that is, terms). In the second stage (or else, the HC part), VEPHC improves the homogeneity and completeness of the generated clusters through split and merge operations that are based on the similarities of all inter-cluster elements. The experimental evaluation of VEPHC on two real-world datasets demonstrates its superior performance over numerous state-of-the-art clustering algorithms in terms of F1 scores and Normalized Mutual Information.**

***Index Terms*—short text clustering, text mining, machine learning, unsupervised learning, clustering, data mining**

I. Introduction

Short texts constitute a popular form of text with numerous diverse uses. Having lengths that range from only a few to several tens of words, they are encountered in a broad variety of applications, including microblogs, entitled entities, news headlines, FAQs, search result snippets, etc. For this reason, short text mining and knowledge discovery from short texts have attracted the attention of multiple researchers. Indicative examples of problems involving short text analysis include named entity recognition, entity resolution, topic modeling, person name disambiguation, NLP tasks (such as opinion mining and sentiment analysis), clustering, and others.

In contrast to normal text collections, short texts suffer from two particularly challenging issues [1], [2]. The first one is data sparseness and concerns the absence of important features (that is, terms) from a portion of the input records. In general, data sparseness renders the problem of clustering considerably more difficult, because it blurs the similarities among the involved entities. The second undesired property is high dimensionality, and originates from the utilization of numerous diverse features (terms, n-grams, skip grams etc.) in short texts. This subsequently leads to a series of side effects which are collectively known as "curse of dimensionality".

Short text clustering is a specialization of the general text clustering problem, which in turn is a specialization of the generic data clustering problem. Similarly to its generalizations, an effective short text clustering algorithm must group its input records by fulfilling two goals: *homogeneity* and *completeness*. The former concerns the inclusion of *only* similar elements within a cluster, whereas the latter verifies that *all* similar elements are grouped into the same cluster.

A portion of the relevant works attempted to confront the problem of sparseness by exploiting external sources of information, like Wikipedia [3] and search engines [4]. Nonetheless, these methods are rather impractical, because the process of fetching information from external sources can be prohibitively expensive.

For this reason, several recent works employed corpus-based methods without requiring the existence of an external source [5], [2]. Other models, such as BTM [6] and Generalized VSM [7], tried to overcome sparseness by capturing and quantifying the relationships between pairs of terms in short texts. The drawback of these approaches is that they ignore the potential correlations between three or more terms. In addition, they disregard the second crucial problem, namely, high dimensionality.

In this article we introduce *VEPHC*, a two stage short text clustering algorithm designed to address both the issues of data sparseness and high dimensionality. The first stage constitutes the VEP part of the algorithm and projects the original feature vectors onto a lower dimensional space, based on a hyper-parameter K. The projection is performed by generating all the $\{1 \dots K\}-$feature combinations of the initial text vectors. These combinations are then treated as candidate cluster labels. In order to identify the most suitable (i.e., dominant) candidate, a score is assigned to each projection vector. The vector scores are computed by a function that incorporates several statistics (e.g., the frequency of a candidate, the IDF of the component terms, etc.), so that it accurately reflects the homogeneity and the completeness of the generated cluster. All documents

whose feature vectors are projected onto the same dimensional space are eventually grouped into the same cluster.

Since the dominant combination consists of only a subset of the features of the initial vectors, it can be considered as a projection of that vector onto a lower dimensional space. Consequently, the problem of high dimensionality is controlled, if not eliminated by this approach. Furthermore, in accordance to the spirit of the aforementioned term co-occurrence models [6], [7], the combination of multiple features also limits the problem of data sparseness.

The second stage of VEPHC consists of a post-processing algorithm that enhances the homogeneity and the completeness of the clusters that were generated by the previous stage. The process is divided in two phases. The first phase is responsible for removing the elements that are prohibitively dissimilar to the rest of the elements of a particular cluster. All removed elements can then be inserted into other, more similar clusters, or form new ones. The similarity of an element with a cluster is determined by its (weighted) cosine similarity with the corresponding clustroid element. In the sequel, the second phase of this stage merges multiple similar clusters into one, more complete cluster. This merging procedure is based on the concepts of the traditional agglomerative clustering.

The rest of this paper is organized as follows: Section II provides a brief discourse on the most successful short text clustering methods. The basic preliminary elements and the notation used in this article are presented in Section III. The details of the proposed algorithm are described in Section IV, followed by the experimental evaluation of Section V. The article concludes in Section VI, which summarizes the contributions and the findings of this work.

II. Related Work

The broad utilization and the importance of short text clustering attracted numerous researchers to study the problem in depth. The works of these researchers enriched the literature with a significant number of state-of-the-art solutions.

A portion of the relevant methods introduce probabilistic models for discovering topics contained within a document collection. In this context, the authors of [8] proposed GSDMM, an iterative Gibbs Sampling algorithm for the Dirichlet Multinomial Mixture. The iterative nature of this model renders it inefficient for large-scale corpora, especially when the number of clusters is large. However, it does not require (exact) prior knowledge of the number of clusters, while it achieves satisfactory clustering performance.

Another topic modeling approach, named BTM (Biterm Topic Model), was introducted in [6]. BTM learns the topics by directly modelling the generation of word co-occurrence patterns (i.e. biterms) in the entire dataset. However, BTM does not take into consideration the correlations among three or more words, and so it misses significant information that derives from extended word co-occurrence.

On the other hand, non-negative matrix factorization (NMF) is a quite common technique for learning topics in text mining tasks. In this context, NMF has been successfully employed in text clustering applications. Two relatively recent methods based on NFM were introduced in [9] and [10], respectively. The former explored term correlation data to tackle the problem of data sparseness. Regarding the latter, the authors proposed a novel term weighting scheme for NMF, derived from the Normalized Cut (Ncut) problem on the term affinity graph.

The research for word co-occurrence patterns in short text document collections led to methods that extended the traditional Vector Space Model. For example, the Generalized Vector Space Model of [7] constructs the term-term correlation matrix by randomly sampling terms with probabilities that are proportional to the lengths of their vectors.

Alternatively, some techniques attempt to address the sparsity problem by employing external sources of information, with the aim of enriching the representation of short texts with additional features. Examples of such sources include Wikipedia [3], WordNet [11], HowNet [12], search engines [4], and others. The most important disadvantage of these approaches is that the communication involved and the subsequent retrieval of relevant information is a very expensive procedure. In addition, some of these external information sources do not provide unlimited access and tight usage restrictions may hold [13]. These drawbacks led to the introduction of several corpus-based methods that use no external information in mining short text data [2], [5].

A portion of the relevant research works employed concept decomposition methods, based on the identification of common concepts shared among similar documents in the corpus. Spherical k-means was among the first algorithms that introduced concept vectors to perform text clustering [14]. Instead of creating concept vectors from the cluster centroids, the method of [1], identifies semantic word communities from a weighted word co-occurrence network.

On a wider scope, [15] introduced a a parameter-free multiview clustering approach for detecting coherent groups in crowd scenes. On the other hand, [16] extended the problem by performing clustering on text streams by introducing an online method which integrates the word occurrence semantic information into a graphical model. Finally, [17] proposed a data clustering algorithm based on the factorization of the projected affinity matrix.

III. Overview and Notation

Let us consider a corpus D composed of n short text documents, $\{d_1, \ldots, d_n\}$. According to the established vector space model, each document $d_i \in D$ can be represented by a vector $\mathbf{x}_i = \{x_{i1}, \ldots, x_{il_i}\}$, where x_{ij} is the weight of the jth term of d_i. On the other hand, l_i denotes the length of d_i in number of terms, in other words, the dimensionality of $\mathbf{x}_i$.

In many applications, the weights x_{ij} are determined by adopting the $tf\text{-}idf$ approach, that is,

$$x_{ij} = tf_{ij} \cdot \log(n/f_j). \tag{1}$$

In this formula, tf_{ij} and f_j represent the number of the occurrences of the jth term in d_i and in the entire corpus,

respectively. By adopting this strategy, the corpus D is finally transformed into the vector space $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$.

The problem of hard clustering concerns the partitioning of the space $\mathbf{X}$ into a number of non-overlapping subspaces $\{C_1, \ldots, C_k\}$, so that each vector $\mathbf{x}_i$ falls into exactly one of these subspaces. The goal of a clustering algorithm focuses on the construction of subspaces that accommodate similar elements. In the context of text clustering, this requirement may concern documents of the same or similar semantics, or documents belonging to the same category or class.

According to the related theory of vector projection, a vector $\mathbf{x}$, initially belonging to the space $\mathbf{X}$, can be projected onto another space $\mathbf{X}'$ by multiplying it with an orthogonal projection matrix P (that is, $P = P^2 = P^{\mathrm{T}}$). For example, to project a three dimensional vector onto a plane one must multiply it with a square 3×3 matrix P that has all its elements equal to zero apart from $p_{11} = p_{22} = 1$.

Vector projection is a common technique for reducing the dimensionality of the underlying feature space in machine learning tasks. It has numerous benefits, such as the reduced space and time complexity of the relevant algorithms. For this reason, a vast amount of research has been devoted to the development of effective dimensionality reduction methods in many problems, including text clustering.

IV. Short Text Clustering with Vector Projection

This section is organized into two parts that present in details the two stages of VEPHC. Hence, Subsection IV-A describes the initial formation of the clusters, whereas Subsection IV-B presents a post-processing cluster refinement stage.

A. Stage 1: Feature Vector Projection (VEP Part)

The first stage of VEPHC (also called as VEP part), is inspired by two recent algorithms for matching product titles, namely [18] and [19]. The core idea is to project the vector representations of all similar documents onto the same vector space. Then, the documents that belong to the same vector space will be eventually grouped into the same cluster.

To achieve this, an integer hyper-parameter K is set. Then, for each input vector $\mathbf{x}_i$ with dimensionality l_i we construct a set $\mathbf{X}'_i$ containing all the possible projections of $\mathbf{x}_i$ with a dimensionality of $1, 2, \ldots, K$ elements. That is, we create all the $\{1, 2, \ldots, K\}$-combinations of the elements of $\mathbf{x}_i$.

For example, consider a document d_i of 4 terms t_1, t_2, t_3, t_4 having weights x_1, x_2, x_3, x_4, respectively. So, the vector representation of d_i will be $\mathbf{x}_i = \{x_1, x_2, x_3, x_4\}$. Now, for the setting $K = 3$, $\mathbf{X}'_i$ will accommodate the following projections of $\mathbf{x}_i$: $\{x_1\}$, $\{x_2\}$, $\{x_3\}$, $\{x_4\}$, $\{x_1, x_2\}$, $\{x_1, x_3\}$, $\{x_1, x_4\}$, $\{x_2, x_3\}$, $\{x_2, x_4\}$, $\{x_3, x_4\}$, $\{x_1, x_2, x_3\}$, $\{x_1, x_2, x_4\}$, $\{x_1, x_3, x_4\}$, and $\{x_2, x_3, x_4\}$.

Each of these projections is considered as a candidate cluster label. The objective now is to identify the most appropriate (namely, the dominant) candidate that will lead into a homogeneous and complete cluster. In this way, all the documents that share a common dominant candidate will be considered as similar and, thus, they will be grouped into the same cluster.

For this reason, each projection $\mathbf{x}'_{ij} \in \mathbf{X}'_i$ is assigned a score value $S_{\mathbf{x}'_{ij}}$ according to the following equation:

$$S_{\mathbf{x}'_{ij}} = \frac{1}{l_{\mathbf{x}'_{ij}}} \log f_{\mathbf{x}'_{ij}} \sum_{\forall x_{ij} \in \mathbf{x}'_{ij}} x_{ij}^K, \tag{2}$$

where $f_{\mathbf{x}'_{ij}}$ is the frequency of the projection vector $\mathbf{x}'_{ij}$ and $l_{\mathbf{x}'_{ij}}$ is its dimensionality. The highest scoring projection $\mathbf{x}^*_i$ is then declared as the dominant projection of $\mathbf{x}_i$ and labels a new cluster. Then, all documents that have dominant projections that lie in the same vector space as $\mathbf{x}^*_i$, are placed into the (same) cluster that has been labeled by $\mathbf{x}^*_i$.

Now let us present the properties of Eq. 2 and justify its form. One conclusion that easily derives from the proposed methodology is that all the documents inside the same cluster share at least those words which are also included in the dominant projection. For example, if $\mathbf{x}_1 = \{x_1, x_2, x_3, x_4, x_5\}$, $\mathbf{x}_2 = \{0, x_2, x_3, x_4, 0\}$, and $\mathbf{x}^*_1 = \mathbf{x}^*_2 = \{x_2, x_4\}$, then it follows that $\mathbf{x}_1$ and $\mathbf{x}_2$ share at least t_2 and t_4.

Apparently, the higher the dimensionality of the dominant projection vector, the more similar documents the corresponding cluster contains. This is because the documents included in that cluster will have more words in common. Equivalently, *the higher the dimensionality of the dominant projection vector, the more homogeneous the corresponding cluster is*. On the other hand, the sparsity of short texts make it difficult for two documents to share numerous words. Therefore, the second conclusion obtained is that *high dimensional dominant projections lead to less complete clusters*.

The existence of the sum in Eq. 2 definitely favors the high dimensional projection vectors. Consequently, it is oriented towards the homogeneity of the clusters. However, this is balanced by placing the dimensionality of the projection vector into the denominator of the first term of Eq. 2; its integration there increases the completeness of the produced clusters. The goal of completeness is also served by $f_{\mathbf{x}'_{ij}}$: it boosts the highly frequent (i.e., contained in many documents) vectors.

Furthermore, notice that a similar strategy must be followed for the selection of the value of K. In general, small values of K lead to complete and inhomogeneous clusters and vice versa. Typical values for K range from 2 to 6. Our experiments showed that larger values of K lead to degraded performance, combined with exponentially increased execution times.

As stated earlier, the documents are commonly transformed into vectors by applying $tf\text{-}idf$ (Eq. 1), a method adopted by this work too. Hence, the combination of Eqs. 1 and 2 leads to the following formula for scoring the projection vectors:

$$S_{\mathbf{x}'_{ij}} = \frac{1}{l_{\mathbf{x}'_{ij}}} \log f_{\mathbf{x}'_{ij}} \sum_{\forall x_{ij} \in \mathbf{x}'_{ij}} \left(tf_{ij} \cdot \log \frac{n}{f_j} \right)^K. \tag{3}$$

Finally, we must note that some relevant works ([2], [8]) criticize the usage of $tf\text{-}idf$ in short text documents since in the vast majority of cases it holds that $tf = 1$. Our experiments verified that, indeed, the omission of tf_{ij} from Eq. 3 has a very small impact on the computed scores.

B. Stage 2: Cluster Refinement (HC Part)

At this point, the dominant projections of the input short text documents have been computed. According to Subsection IV-A, each dominant projection corresponds to a cluster that contains all the documents with projections lying in the same vector space. Next, we introduce the second stage of VEPHC, which is designed to further improve the homogeneity and the completeness of the generated clusters.

Before we proceed with the description of the algorithm, we present the necessary notation. So, the universe that accommodates all the clusters is denoted by C and $c \in C$ is a member of C. A candidate cluster for accepting a new element $\mathbf{x}$ is denoted by c', whereas c^* is the most similar (i.e., less distant) cluster to $\mathbf{x}$. We also require a special cluster $\mathcal{C}$ for temporarily storing the deleted elements. Finally, n_c is the maximum number of elements contained in a cluster of C.

In addition, this stage requires the computation of the similarity between an element $\mathbf{x}$ and a cluster c. The relevant literature contains various methods for this purpose, including the simple and complete linkage, the Euclidean distance from the centroid of c, the Ward method, etc. In this work, we utilized the clustroid $\mathbf{u}_c$; that is, the element of c that has the maximum similarity with the rest of the elements of c. In contrast to centroids, the clustroids are real elements and have been proved more effective in text clustering applications.

The cluster refinement stage depends on two similarity thresholds. The first one is called the *homogeneity threshold* T_h and determines whether an item should be removed from a cluster or not. As its name suggests, T_h is a factor that regulates the homogeneity, since it decides for the eviction of dissimilar elements from a cluster. On the other hand, T_c is the *completeness threshold* and indicates the appropriateness of an item for being inserted into a cluster. It also controls the merging process of two similar clusters into a larger one.

The HC Part of VEPHC is divided into two phases. The first phase transfers elements from one cluster to another, more similar one. If this is not possible, a new cluster is created and the deleted element is moved there. After this operation, the second phase merges highly similar clusters into a larger and more complete cluster.

1) Element Transfer and Creation of New Clusters: The basic operations of phase 1 are presented in Algorithm 1. The process begins with the initialization of the special cluster $\mathcal{C}$ that will temporarily handle the deleted elements. The clustroid for each cluster is also computed at this point (steps 2–4).

The purpose of steps 5–25 is to detect elements that are loosely connected to their clusters. Such elements are immediately deleted from their clusters and moved to a more suitable cluster, if one exists. Hence, this part aims at improving both the completeness and the homogeneity of the output clusters.

More specifically, the process iterates through all elements of all clusters of C. For each item $\mathbf{x} \in c$ the similarity with the clustroid $\mathbf{u}_c$ is computed. If this similarity value $T_{\mathbf{x},\mathbf{u}_c}$ is smaller than T_h, then the item is considered irrelevant to c and it is evicted from it (steps 7–9). Apparently, this strategy improves the homogeneity of c.

Algorithm 1: HC Part, Phase 1: Element Transfer and Creation of New Clusters

```
1  initialize an empty cluster C;
2  for each cluster c ∈ C do
3  |  u_c ← clustroid of c;
4  end
5  for each cluster c ∈ C do
6  |  for each element x ∈ c do
7  |  |  T_{x,u_c} ← similarity between x and u_c;
8  |  |  if T_{x,u_c} < T_h then
9  |  |  |  remove x from c;
10 |  |  |  T_max ← 0, c* ← NULL;
11 |  |  |  for each cluster c' ∈ C do
12 |  |  |  |  T_{x,u'_c} ← similarity between x and u'_c;
13 |  |  |  |  if T_{x,u'_c} > T_max then
14 |  |  |  |  |  T_max ← T_{x,u'_c};
15 |  |  |  |  |  c* ← c';
16 |  |  |  |  end
17 |  |  |  end
18 |  |  |  if T_max ≥ T_c then
19 |  |  |  |  c* ← c* ∪ {x};
20 |  |  |  else
21 |  |  |  |  C ← C ∪ {x};
22 |  |  |  end
23 |  |  end
24 |  end
25 end
26 C_new ← ∅;
27 for each element x ∈ C do
28 |  remove x from C;
29 |  T_max, c* ← perform steps 11–17 on C_new;
30 |  if T_max ≥ T_c then
31 |  |  c* ← c* ∪ {x};
32 |  else
33 |  |  create and initialize new cluster c_new ← ∅;
34 |  |  c_new ← c_new ∪ {x}, u_{c_new} ← x;
35 |  |  C ← C ∪ {c_new}, C_new ← C_new ∪ {c_new};
36 |  end
37 end
```

Next, a search for the most similar cluster c^* is performed. The loop in the steps 11–17 treats all clusters as candidates and identifies the candidate c^* that has the maximum similarity $T_{\max}$ with $\mathbf{x}$. If $T_{\max} \geq T_c$, then the document is highly related to c^* and it is transferred there. In the opposite case, it is inserted in the special cluster $\mathcal{C}$ for further processing. Notice how this approach improves the completeness of c^*.

Now the issue is to determine the destiny of the elements of $\mathcal{C}$ that have been left without a cluster. Although these elements match none of the existing clusters, they could match each other. The steps 26–37 show that an evicted item may lead to the creation of a new cluster. However, before this is done, we firstly check if it matches any of the newly created clusters C_{new}. If it does, we move it to its most similar cluster.

Algorithm 2: HC Part, Phase 2: Cluster Merging

1 **for** *each cluster* $c \in C$ **do**
2 | $\mathbf{u}_c \leftarrow$ clustroid of c;
3 **end**
4 $sim\_array[*] \leftarrow 0$, $msc\_array[*] \leftarrow -1$;
5 **for** *each cluster* $c \in C$ **do**
6 | $msc\_array[c] \leftarrow$ most similar cluster (MSC);
7 | $sim\_array[c] \leftarrow$ similarity with MSC;
8 **end**
9 **while** *merge* **do**
10 | $merge \leftarrow$TRUE;
11 | $c' \leftarrow$NULL, $c'' \leftarrow$NULL;
12 | **for** *each cluster* $c \in C$ **do**
13 | | $T_{\max} \leftarrow 0$;
14 | | **if** $sim\_array[c] > T_{\max}$ **then**
15 | | | $T_{\max} \leftarrow sim\_array[c]$;
16 | | | $c' \leftarrow c$, $c'' \leftarrow msc\_array[c]$;
17 | | **end**
18 | **end**
19 | **if** $T_{\max} > T_c$ **then**
20 | | $merge \leftarrow$TRUE;
21 | | $c' \leftarrow c' \cup c''$;
22 | | $\mathbf{u}'_c \leftarrow$ clustroid of c';
23 | | $msc\_array[c'] \leftarrow$ MSC;
24 | | $sim\_array[c'] \leftarrow$ similarity with MSC;
25 | | $C \leftarrow C - \{c''\}$;
26 | | update msc_array, sim_array;
27 | **end**
28 **end**

In the opposite case, a new cluster c_{new} is created and $\mathbf{x}$ is transferred there. The new cluster is inserted to the universe C, whereas $\mathbf{x}$ becomes its clustroid. Therefore, c_{new} will be a candidate during the processing of the next elements of $\mathcal{C}$.

Finally, the worst-case time complexity of Algorithm 1 is $O(|C|n_c^2 + |C|n)$. The first term concerns the computation of clustroids (steps 2–4), whereas the second term derives from the two loops of steps 5–25 and 26–37.

2) Cluster Merging: As stated earlier, during the second phase the highly similar clusters are merged into larger and more complete clusters. This is performed in a spirit that is similar to the traditional agglomerative clustering method. Algorithm 2 summarizes the basic steps of the operation.

Initially, the clustroids are computed for all clusters (steps 1–3). In the sequel, two auxiliary arrays are created and initialized. Namely, msc_array and sim_array store in their ith element the most similar cluster of c_i and the value of the similarity between their clustroids, respectively.

These arrays will be employed later in the cluster merging loop (steps 9–28). The execution flow is controlled by a boolean variable $merge$ that is immediately set to false at the beginning of each iteration. If a merging between two clusters is possible during the current iteration, $merge$ becomes equal to true to allow the execution of the next iteration. On the other hand, if no similarity exceeds T_c, $merge$ remains false and the merging process is terminated at this point.

Now, inside the loop, the most similar pair of clusters c' and c'' is identified at steps 11–18. If their similarity exceeds the aforementioned completeness threshold T_c, then c' and c'' are merged. This is performed by transferring all the elements of c'' to c' and subsequently deleting c''. Finally, the new clustroid of c' is computed, and its most similar cluster is stored in the corresponding position of msc_array.

The worst-case time complexity of Algorithm 2 is upper bounded by $O(|C|n_c^2 + |C|^2)$. In details, $O(|C|n_c^2)$ for steps 1–8, plus $O(|C|)$ for step 4, plus $O(|C|^2)$ for steps 5–8, plus $O(|C|(|C| + n_c^2))$ for the merging loop (steps 9–28).

V. Experiments

This section presents the experiments that highlight the usefulness of the proposed method. It is organized in 4 parts that describe: i) the employed datasets (Subsection V-A), ii) the utilized performance evaluation measures (Subsection V-B), iii) the effects of the hyper parameter K in the performance of VEPHC (Subsection V-C), and iv) the results of the performance evaluation against numerous clustering methods.

A. Datasets

The evaluation process was performed by utilizing two real-world experimental datasets. The first one[1], (abbrev. *PRUN*), consists of 35311 product titles that were acquired by manually crawling PriceRunner[2], a popular online product comparison platform. The titles in this dataset are grouped in 13233 clusters of similar products that match each other. The average and maximum title lengths are 8.23, and 41 terms, respectively.

The second dataset, named *UNA*, is a subset of the UCI News Aggregator dataset[3] that we created by random sampling. This was done to allow some slow methods (e.g., Agglomerative) to be executed within acceptable times. This subset comprises 50138 titles of news articles, grouped in 823 distinct stories (i.e., clusters). The titles of the news articles were shorter than those of the products, since they included on average 7 terms, whereas the longer among them had 15 terms. Table I summarizes the characteristics of these two datasets.

TABLE I
Experimental datasets

	n	$\|C\|$	$l_{\max}$	l_{ave}
PriceRunner (PRUN)	35311	13233	41	8.23
UCI News Aggregator (UNA)	50138	823	15	7.05

B. Evaluation Measures

The performance of the compared algorithms was measured by calculating the values of $F1$ and NMI (Normalized Mutual Information). Both of them are commonly utilized in the relevant literature for evaluating clustering methods.

[1]https://www.kaggle.com/lakritidis/product-classification-and-categorization

[2]https://www.pricerunner.com/

[3]https://www.kaggle.com/uciml/news-aggregator-dataset

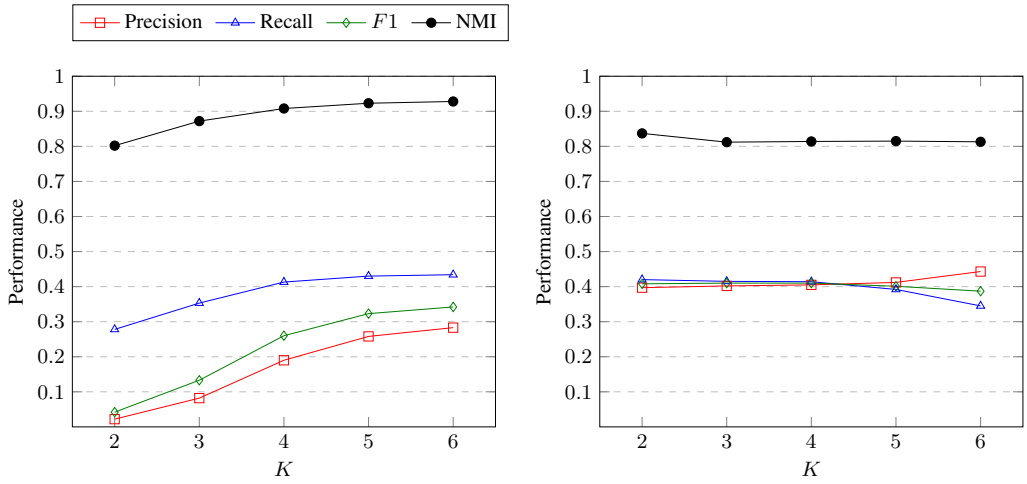

Fig. 1. Performance fluctuation of VEPHC (without the verification stage) against varying values of the hyper parameter K, on the PriceRunner (left), and UCI News Aggregator (right) datasets.

More specifically, $F1$ is a score that determines the clustering accuracy and is given by the formula $F1 = 2\mathcal{PR}/(\mathcal{P}+\mathcal{R})$, where $\mathcal{P}$ and $\mathcal{R}$ denote Precision and Recall, respectively. The computation of $\mathcal{P}$ and $\mathcal{R}$ was performed by firstly creating all pairwise matches between the elements of all constructed clusters. In the sequel, the created matches were compared against the ground truth matches, thus allowing the calculation of the number of the True/False Positive/Negative entries.

On the other hand, NMI measures the mutual information shared by the cluster assignments C of the algorithm in question and the ground truth clusters C_T:

$$NMI(C_T, C) = 2\frac{I(C_T; C)}{H(C_T) + H(C)} \tag{4}$$

where $H(C_T)$ and $H(C)$ are the entropies of the labelled and clustered sets respectively, and $I(C_T; C) = H(C) - H(C|C_T)$ is the mutual information shared between C and C_T.

C. The impact of K in the Performance of VEPHC

As stated earlier, K is a crucial hyper parameter of VEPHC, since it determines the upper bound of the dimensionality of the produced projection vectors. Figure 1 illustrates the impact of K in the overall performance of the proposed algorithm. The performance fluctuations on the PRUN and UNA datasets are depicted on the left and right diagrams, respectively. Each diagram contains the plots of the four aforementioned evaluation metrics, namely, Precision, Recall, $F1$ and NMI. In order to accurately estimate the impact of K, we report the performance of the VEP part of the proposed algorithm only.

On the PRUN dataset, the values of all four metrics were maximized when $K = 6$. In particular, $F1$ and NMI were measured equal to 0.323 and 0.928, respectively. On the other hand, for $K = 2$, the Precision was very small and this directly affected the value of $F1$. NMI was roughly equal to 0.802 for this setting. The plotted curves show that all measures were progressively improved as the value of K was increased.

In contrast, on the UNA dataset the variations in the value of K led to only slight performance changes. Therefore, the impact of K in this case was smaller than it was on the PRUN dataset. This is especially true for values of $K \geq 3$, where all four evaluation metrics are modified by only infinitesimal margins. Another difference is that here, $F1$ and NMI were maximized for $K = 2$. More specifically, their values were found roughly equal to 0.408 and 0.837, respectively. This observation reveals that the best projections of the initial feature vectors were the two-dimensional ones and correspond to plain pairs of words (that is, bigrams).

The discrepancies between the performance differences on the two datasets lead to the remarkable inference that the ideal value of K depends on the number n of input documents vs. the number $|C|$ of the generated clusters. Equivalently, the ideal K value depends on the ratio $n/|C|$.

Particularly, if $n/|C|$ is high, then the number of clusters is relatively small, which also means that these few clusters will accommodate numerous data points. Thus, according to the discussion of Subsection IV-A, this situation is better served by small values of K, since this setting creates very low-dimensional projector vectors that correspond to few, complete, and inhomogeneous clusters. On the contrary, if $n/|C|$ is small, then, obviously, the number of the created clusters is high. This means that they contain fewer data points, making them more homogeneous and less complete.

D. Performance Evaluation

In this subsection we compare the proposed VEPHC method with a series of existing state-of-the-art clustering techniques. More specifically, we implemented four generic data clustering algorithms, including DBSCAN [20], Leader Clustering [21], Agglomerative Clustering, and k-Means. The first three among them require the setting of a similarity threshold (as a hyper parameter) that determines whether two clusters are similar enough to be merged, or an element is similar to a cluster. We executed each of these three methods ten times with different values of the similarity threshold and we report the best run. Regarding k-Means, we used random initialization and we set

TABLE II
PERFORMANCE EVALUATION OF VARIOUS CLUSTERING ALGORITHMS ON THE PRICERUNNER AND UCI NEWS AGGREGATOR DATASETS

Method	Setup	PriceRunner			UCI News Aggregator		
		$F1$	NMI	Time	$F1$	NMI	Time
VEPHC	$K_1 = 6, K_2 = 2$	0.385	0.946	305.1	0.471	0.851	10.2
k-Means	$k = \|C\|, I = 10$	0.048	0.399	494.2	0.028	0.697	104.3
Agglomerative	–	0.314	0.935	166.7	0.454	0.835	2492.8
Leader Clustering	–	0.306	0.934	17.2	0.284	0.778	17.1
DBSCAN	$minPoints = 2$	0.055	0.425	58.4	0.256	0.790	130.5
GSDMM-1	$\alpha = \beta = 0.1, k = 1.5 \cdot \|C\|$	0.002	0.404	3246.2	0.171	0.753	390.4
GSDMM-2	$\alpha = 0.001, \beta = 0.01, k = 1.5 \cdot \|C\|$	0.008	0.631	3317.3	0.424	0.820	453.9
Spherical k-Means	$k = \|C\|, I = 10$	0.226	0.903	426.1	0.335	0.757	119.6
vk-Means	$k = \|C\|, I = 10$	0.220	0.900	484.4	0.154	0.614	104.7
Cosine similarity	$tf - idf$ weights	0.248	–	31.3	0.388	–	75.2
Jaccard Index	$tf - idf$ weights	0.237	–	31.5	0.388	–	74.9

k equal to the real number of the clusters, that is, $k = |C|$. In DBSCAN, the $minPoints$ parameter that determines whether a data point is an outlier (noise) or not, was set equal to 2.

We also developed three text clustering methods, namely, GSDMM [8], Spherical k-Means [1], [14], and another variant of k-Means (which we named vk-Means), where the centroids and the Euclidean distances of the original algorithm have been replaced by clustroids and weighted cosine similarity, respectively. We used two settings for the α and β hyper parameters of GSDMM: the first one is $\alpha = \beta = 0.1$ and complies with the suggestions mentioned in [8] and [22]. Nevertheless, we discovered that the setting $\alpha = 0.001, \beta = 0.01$ that was utilized in a public implementation[4] yielded considerably higher performance. Additionally, GSDMM requires an estimation of the real number of clusters. This estimation must be greater than the real number of clusters. Therefore, we set $k = 3|C|/2$. Regarding the number of iterations, in all iterative methods we adopted the selection of [1] and we set it equal to 10.

Moreover, we report results from the direct application of two popular string similarity measures, namely, cosine similarity and Jaccard index. Although they do not constitute hard clustering methods (in fact, no clusters are constructed), we considered their comparison with VEPHC interesting. In both cases, the plain token counts have been replaced by tf-idf scores. Similarly to the aforementioned data clustering methods, we executed both metrics 10 times, by setting different values for the similarity threshold each time.

All the attested algorithms have been implemented in C++ and have been included in SHTECLib[5], a short text clustering library that was developed for the requirements of this work. The code was executed on a machine equipped with an Intel CoreI7-7700 and 32GB of RAM, running Linux Mint 19.03.

The results of the performance evaluation of the compared algorithms are presented in Table II. The second column briefly reports the values of the various hyper parameters of each reported method. VEPHC outperformed all the adversary approaches in terms of $F1$ and NMI on both datasets. More specifically, in the PRUN dataset VEPHC achieved $F1 = 0.385$ and $NMI = 0.946$. According to Figure 1, this performance was measured for $K = 6$. Regarding the refinement stage, multiple values of T_h and T_c led to similar results, e.g. $0.4 \leq T_h \leq 0.5$ and $0.8 \leq T_c \leq 0.9$.

The method that followed VEPHC in performance was agglomerative clustering; its $F1$ and NMI scores were 0.320, and 0.936, respectively. In other words, VEPHC outperformed its strongest adversary by more than 20% in terms of $F1$, and more than 1% in terms of NMI. The accuracy of Leader Clustering was also close to the one achieved by Agglomerative Clustering, that is, $F1 = 0.306$ and $NMI = 0.934$.

On the other hand, k-Means and DBSCAN definitely failed on effectively clustering the products of PRUN, even though the former was supplied with the correct number of clusters. The measured values of their $F1$ and NMI scores were considerably lower than 0.1 and 0.45, respectively.

Regarding the family of the examined text clustering algorithms, Spherical k-Means and vk-Means had similar performances, with the first being infinitesimally more effective. Nevertheless, they were both outperformed by VEPHC by huge margins, ranging from 70% to 75%, respectively, in terms of $F1$ scores. However, their achieved NMI scores were significantly better: 0.903 and 0.9, respectively.

Similarly to k-Means and DBSCAN, GSDMM also failed in this task in both hyper parameter settings. Although the relevant literature has highlighted the usefulness of this model, in this particular case, it was the weakest method among all. Nevertheless, in our second dataset its performance was substantially improved and, under the setting $\alpha = 0.001, \beta = 0.01$, it became the third most effective approach. In contrast, under the setting $\alpha = 0.1, \beta = 0.1$ that was suggested in [8], the performance of GSDMM was not satisfactory.

Regarding the UNA dataset, VEPHC was again the most effective algorithm, achieving $F1 = 0.471$ and $NMI = 0.851$. These values were obtained by setting its hyper parameters to $K = 2, T_h = 0.2$, and $T_c = 0.3$. This reveals that VEPHC requires some fine tuning before it is effectively applied in datasets with different characteristics.

Similarly to the previous case, the second algorithm in the performance ranking was Agglomerative Clustering ($F1 = 0.454, NMI = 0.834$), followed by Spherical k-Means

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

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

($F1 = 0.335, NMI = 0.757$) and Leader Clustering ($F1 = 0.284, NMI = 0.778$). Other significant observations include the low accuracy of k-Means and vk-Means, as well as the enhancement of the clustering quality of DBSCAN. Apparently, density-based clustering worked better on this case.

In Table II we also present indicative running times of the compared algorithms. As stated earlier, the performance of the algorithm is significantly affected by the value of the hyper parameter K. Indeed, in the case of UNA dataset (where $K = 2$), VEPHC was the fastest method among all. Regarding the other methods, Leader Clustering was about 1.7 times slower; GSDMM, all the variants of k-Means and DBSCAN were outperformed by more than one order of magnitude.

On the PRUN dataset ($K = 6$), the execution duration of VEPHC was considerably larger. In particular, Leader, Agglomerative, and DBSCAN were faster than the proposed method. The other methods were again outperformed by VEPHC. As an indication of the effect of K in the overall running times, we made measurements for $K = 2, 3, 4$, and 5, which were 34.5, 30.3, 36.7, and 85.1 seconds, respectively.

Obviously, in the first three cases the overall time is dominated by the HC part. For $K \geq 5$ the situation is reversed and the creation of the projection vectors becomes the main bottleneck. For $K = 5$, the performance of VEPHC was $F1 = 0.364$ and $NMI = 0.935$, that is, it still outperformed all its adversary methods. Notice that with this setting the execution is more than 3.5 times faster than it was with $K = 6$; consequently, it constitutes an attractive alternative.

VI. Conclusions

In this paper we introduced VEPHC, a two-stage clustering algorithm for short text documents. The proposed method is designed to limit the native sparseness and high dimensionality of short texts, whereas it attempts to improve the homogeneity and completeness of the generated clusters.

Initially, for each input text vector, VEPHC constructs a set of low-dimensional projection vectors that include variable-sized combinations of K features at most. The projections are then assigned a score by a function that focuses on cluster homogeneity and completeness. The projection that achieved the highest score becomes the label of a new cluster. The documents whose projection vectors lie in the same feature space are also inserted into the corresponding cluster. During the second stage, a refinement mechanism further improves the homogeneity and completeness by transferring elements from one cluster to another and by merging similar clusters.

VEPHC was evaluated against numerous state-of-the-art methods by using two real-world datasets, and was found superior in terms of clustering performance. In particular, it outperformed 4 generic and 3 text clustering methods in terms of both $F1$ and NMI scores.

The main conclusion of this work is that the projection of the initial feature vectors onto a lower dimensional space is a promising technique in short text clustering applications. Further benefits also derive by carefully transferring elements from one cluster to the other, or by merging similar clusters.

References

[1] 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.

[2] C. T. Zheng, C. Liu, and H. San Wong, "Corpus-based topic diffusion for short text clustering," *Neurocomputing*, vol. 275, pp. 2444–2458, 2018.

[3] 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*, 2007, pp. 787–788.

[4] 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*, 2006, pp. 377–386.

[5] D. Pinto, P. Rosso, and H. Jiménez-Salazar, "A self-enriching methodology for clustering narrow domain short texts," *The Computer Journal*, vol. 54, no. 7, pp. 1148–1165, 2011.

[6] 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*, 2013, pp. 1445–1456.

[7] 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*, 2015, pp. 805–810.

[8] J. Yin and J. Wang, "A Dirichlet multinomial mixture model-based approach for short text clustering," in *Proc. of the 20th ACM Conference on Knowledge Discovery and Data Mining*, 2014, pp. 233–242.

[9] 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*, 2013, pp. 749–757.

[10] 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*, 2012, pp. 2259–2262.

[11] 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*, 2009, pp. 919–928.

[12] L. Wang, Y. Jia, and W. Han, "Instant message clustering based on extended Vector Space Model," in *2nd International Symposium on Intelligence Computation and Applications*, 2007, pp. 435–443.

[13] D. Milne, O. Medelyan, and I. H. Witten, "Mining domain-specific thesauri from Wikipedia: A case study," in *IEEE/WIC/ACM International Conference on Web Intelligence*, 2006, pp. 442–448.

[14] I. S. Dhillon and D. S. Modha, "Concept decompositions for large sparse text data using clustering," *Machine Learning*, vol. 42, no. 1-2, pp. 143–175, 2001.

[15] Q. Wang, M. Chen, F. Nie, and X. Li, "Detecting coherent groups in crowd scenes by multiview clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 1, pp. 46–58, 2018.

[16] 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*, 2020, pp. 766–776.

[17] M. Chen, Q. Wang, and X. Li, "Adaptive projected matrix factorization method for data clustering," *Neurocomputing*, vol. 306, pp. 182–188, 2018.

[18] L. Akritidis, A. Fevgas, P. Bozanis, and C. Makris, "A self-verifying clustering approach to unsupervised matching of product titles," *Artificial Intelligence Review*, pp. 1–44, 2020.

[19] 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*, 2018, pp. 1–10.

[20] 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*, 1996, pp. 226–231.

[21] H. Spath, *Cluster analysis algorithms for data reduction and classification of objects*. Ellis Horwood Chichester, 1980.

[22] J. Yin and J. Wang, "A text clustering algorithm using an online clustering scheme for initialization," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 1995–2004.