Scalable Overlapping Co-Clustering of Word-Document Data

Fabrício Olivetti de França
Federal University of ABC (UFABC) – Center of Mathematics, Computing and Cognition (CMCC)
R. Santa Adélia 166, CEP 09210-170, Santo André, Brazil
Email: folivetti@ufabc.edu.br

Abstract—Text clustering is used on a variety of applications such as content-based recommendation, categorization, summarization, information retrieval and automatic topic extraction. Since most pair of documents usually shares just a small percentage of words, the dataset representation tends to become very sparse, thus the need of using a similarity metric capable of a partial matching of a set of features. The technique known as Co-Clustering is capable of finding several clusters inside a dataset with each cluster composed of just a subset of the object and feature sets. In word-document data this can be useful to identify the clusters of documents pertaining to the same topic, even though they share just a small fraction of words. In this paper a scalable co-clustering algorithm is proposed using the Locality-sensitive hashing technique in order to find co-clusters of documents. The proposed algorithm will be tested against other co-clustering and traditional algorithms in well known datasets. The results show that this algorithm is capable of finding clusters more accurately than other approaches while maintaining a linear complexity.

Keywords—co-clustering; text clustering; hashing

I. INTRODUCTION

The advent of the world wide web (WWW) generated a huge amount of textual data in web pages, discussion boards, social networks and chats. This data is usually unstructured in the sense that it does not have well specified fields identifying several of its characteristics like topic, cited persons, keywords and sentiment or intent of the text. Such property lead to an interest in creating specific machine learning techniques to deal with such data. These techniques compose the field of text mining and they are applied to a varied set of problems [1], [2], [3], [4].

One of such problems, named text clustering, tries to find groups of texts that are similar regarding a definition of similarity concerning the objective of the research. One difference that makes text clustering distinct from data clustering is that each object (i.e., a text document) contains only a small fraction of the set of features from the text corpora. Because of this sparseness, two texts belonging to the same topic may have just some few words in common. So the definition of similarity is by itself considered a challenge to this problem. A textual dataset can also be described as a set of pairs \((d, w)\) indicating that the document \(d\) contains the word \(w\).

Another characteristic of this type of data is the lack of a fixed set of attributes. If you have a corpus with million of documents, not only the object set is already intractable, but the feature set will also expand to thousands of words making it intractable to most clustering algorithms due to the curse of dimensionality.

One way to reduce the size of the dataset is by using a feature selection technique capable of finding the most important words of the corpus. For this task it is usually performed a supervised mutual information [5] of the feature space in order to find the words that gives the most information regarding the corpus. The downside of this approach is that you must know, or determine, a priori the grouping configuration of the document data (or a part of it) in order to apply this method.

Another possible approach is the unsupervised word selection called Partitioning Around Medoids (PAM) [6]. But this technique is reported to not give a very good set of features, reducing the accuracy of the text classification task [7].

One technique that can deal with these problems in a natural way, and without the requirement of prior knowledge of the dataset is the co-clustering [1], [2], [3], [4]. A Co-Cluster of a textual data is a subset \(D'\) of documents and a subset \(W'\) of words with the constraint that each pair \((d', w')\), for all \(d' \in D'\) and \(w' \in W'\), belongs to the data set (hard co-cluster). Sometimes this constraint is relaxed to just a percentage of these pairs (soft co-cluster).

So a Co-Cluster of text data would group together a number of documents that share a subset of words in common. This leads to two important properties; i) each group is evaluated using only a smaller subset of features selected exclusively for this group; ii) a given object or a given feature may belong to more than one group (overlapping co-clusters). The first property solves the sparseness of the data and, the practical aspect of this second property is that a given document may belong to different groups if it discuss more than one topic or, a given word may be used to define different groups in different contexts when used with combination of different words from the set.

Two notable application of co-clustering to text mining [1], [2] applies non-overlapping hard co-clustering methodology. In each of these papers the authors reported results with higher accuracy than other well-known algorithms for this same problem. But the use of overlapping soft co-clusters may improve such results by removing the constraint
that each document and each word correspond to only one topic and that, in order for a document to belong to a topic, it must contain a hard defined set of words. Also, these works require that the number of clusters of object and clusters of features are defined a priori. In this paper I will propose an approximate Co-Clustering algorithm that is capable of finding co-clusters in dyadic data allowing overlapping of clusters and without the need to set the number of clusters a priori.

The paper is organized as follows: section II will describe the proposed algorithm in details and its properties, in section III some experiments on real world text data is performed and analyzed, finally, the section IV concludes this work and give some insights for future works.

II. HASH BASED LINEAR CO-CLUSTER

The proposed Co-Clustering algorithm, named Hash Based Linear Co-Cluster (HBLCoClust) is composed of three steps. In the first step, a Locality-sensitive Hashing algorithm [8] is used to associate each object with $n$ possible co-clusters through a hash-key. Two or more objects will have an equal hash-key with an associated probability given they have at least $p$ features in common. After this step, the objects with the same hash-key are associated to a candidate co-cluster.

In the second step, for each candidate co-cluster it is iteratively inserted each feature that is associated to a percentage of this co-cluster’s objects. After this step, the feature set for this co-cluster is defined. In the third and final step, it is iteratively inserted each object that is associated to a percentage of this co-cluster’s features.

As a first property, it should be noted that this algorithm can find soft co-clusters increasing the size of the object and feature sets without degrading the information contained in this co-cluster, depending on the defined percentage.

These steps will be described in more details in the following subsections.

A. Step One: Locality-sensitive Hashing

The technique known as Locality-sensitive Hashing (LSH) is a method capable of clustering high-dimensional data using hash functions. This method uses the idea that if a hashing function is weak to collision, then two similar objects will be allocated to the same bucket. If we have a probability of collision associated with how similar two objects are, we can simplify a clustering procedure by just applying this hash function to each object and each cluster will be defined by each bucket of the hashing procedure.

A common similarity measure used in LSH is the Jaccard Index that calculates the ratio between the number of features two objects have in common and the number of features belonging to one or both the objects:

$$J(o_1, o_2) = \frac{|F_1 \cap F_2|}{|F_1 \cup F_2|}. \quad (1)$$

If we define a function $\pi(f)$ that returns a number representing the position of $f$ in a random permutation of the features set and define a hash function $h(o) = \min \pi(f)$, then the probability of two objects colliding into the same bucket is given by the Jaccard coefficient. So, by generating $p$ different permutations, applying the hash function to each object, and repeating the experiment $n$ times, the probability of two objects colliding into at least one bucket is:

$$P_{\text{collision}}(o_1, o_2)_{p,n} = 1 - (1 - J(o_1, o_2)^p)^n. \quad (2)$$

Although the generation of $n \times p$ different random permutations of the features set can have a high computational cost, it is possible to use a function that approximates these permutations called Min-wise independent permutation [9] such as $f(x) = a \times x + b \mod P$ where $x$ is the feature index, $a$ and $b$ are random integers and $P$ is a large prime number. With this function, each permutation is defined by different values of $a$ and $b$, and whenever $f(x_1) < f(x_2)$ it means that $x_1$ appears before $x_2$ in this given permutation.

So, in order to apply this algorithm with a text corpus, first each word of the corpus must be associated to a unique integer index (by means of a string hashing function). Then, it must be generated $l = n \times p$ different values for $a$ and $b$, creating $l$ permutation functions. For each document, each of the $l$ functions are applied to its words and the minimum value is stored. Finally, a vector of size $l$ is generated storing the feature that has the minimum value for each corresponding permutation function. After this step, the values of this vector is grouped by each $p$ values that, when composed together as a string, generates a single LSH hash string.

The complexity of this step is given by the application of each hash function into each feature for each object. Since on a document corpus each document has an average of $k$ words, the complexity is $O(l.k.n)$, with $l$ and $k$ constant, and thus being linear to the number of documents.

B. Step Two: Co-cluster Feature Set

In the next step, for each candidate co-cluster, composed of a set of documents, it is investigated which features these documents have in common by following these steps: i) create a hash map counting the occurrence of words belonging to each document of the Co-Cluster; iii) for each word of the hash map, if its counter is higher than a percentage of the number of documents in the candidate co-cluster, insert it into the co-cluster feature set.

It is easy to see that this step has complexity $O(k.n' + k)$, with $n'$ as the number of documents in the candidate set and $k$ the average number of words per document, for each candidate, summing up to $O(k.n)$ when applying this step to each co-cluster candidate.
C. Step Three: Co-cluster Object Set

In the final step, the above procedure is repeated but hashing the documents that contains each feature of the co-cluster candidate instead. The complexity for this step is similar to the same as the last step: \(O(k.n' + n')\), and summing up to \(O(k.n)\).

At steps two and three, any co-cluster that does not meet a minimum number of objects (\(\text{min\_rows}\)) and minimum number of features (\(\text{min\_cols}\)) requirements, are eliminated from the candidate set.

After the completion of these three steps, each co-cluster will be composed of a set of objects and a set of features such as the features will be present in \(\tau\%\) of the documents.

One thing to notice on this approach is that, differently from other co-clustering algorithms, HBLCoClust does not guarantee 100\% coverage of the entire data set, this means that not every object and feature from the original set will belong to one co-cluster from the set.

Another property of the proposed approach is that, in contrast with others co-clustering approaches applied to text mining, HBLCoClust outputs every co-cluster it finds, experimentally this lead to thousands of unique co-clusters extracted from textual datasets. So, in order to compare with traditional approaches an additional step was applied to the result of HBLCoClust such as to create exactly \(k\) document clusters.

D. Generating \(k\) Clusters

This step consists of first generating a graph from the extracted co-clusters, this is done by representing each object as a node and each edge representing that two objects belongs to the same co-cluster. Surprisingly, during the experimental setup, this leads to an almost completely connected graph, with just very few nodes disconnected. After the graph is built, a graph partitioning algorithm, called METIS [10], is applied generating \(k\) different clusters.

III. Experiments

The experimental setup faithfully replicate the ones in [7], for this reason the table depicting the obtained results on the aforementioned paper will be reproduced here. This follows the same procedures adopted by previous authors in [1], [2], [7]. For this purpose it was used different subsets from the well-known 20-Newsgroup dataset (NG20) that consists of 19,997 messages received on 20 newsgroups of distinct subjects. Each message are preprocessed by removing the subject lines and stop-words.

In [1], [2], [7] it was also applied a feature selection based on supervised mutual information (SMI) in order to select the 2,000 most significant words. As stated in [7], not only this makes the text clustering needs less computational effort, but also helps the algorithms to improve their results. In practice, if there is no information about the different classification of the textual data this approach cannot be applied, so in [7] it was also made an experiment by selecting the top words with an unsupervised feature selection algorithm as well.

Since the proposed approach already performs an unsupervised and local feature selection, the HBLCoClust will work with the full text data, without the use of any external feature selection algorithm and it will be compared with the results reported in [7] using SMI, even though this may give advantage to the former.

It was created 10 different samples extracted from 6 distinct subsets of documents from the dataset. Each subset is composed of \(#\_docs\) sampled documents regarding \(#\_classes\) different classes (Table I).

The results comparisons are performed against 5 other algorithms: Agglomerative Hierarchical Clustering using LSA [11], \(\chi\text{-Sim}\_p\) (co-cluster based similarity) with supervised feature selection (SMI) and unsupervised feature selection (PAM) [7], and the co-clustering algorithms ITCC [1] and BVD [2]. Notice that every other algorithm of this list used the SMI feature selection beforehand.

It was used the micro-averaging precision measure [1] in order to compare the accuracy of the classification algorithms and the reported result is the average of such measure over the 10 trials. The parameters used for each algorithms are the same reported in [7].

The parameters used for HBLCoClust were the following: \(n = 10,000, p = 3, \text{min\_rows} = 2, \text{min\_cols} = 2\) and the hash function prime number 109, 297 for every dataset. The parameter \(\tau\) was set as 86\% for M2 dataset and 100\% for every other dataset.

The results are described in table II. From this table it is possible to see that HBLCoClust obtained the best results in every dataset except one, M5, in which it was bested only by \(\chi\text{-Sim}\_p(SMI)\) with a difference of 1\%. In every other dataset, HBLCoClust improved the results from 3\% to 10\% when compared with the second best results.

It is also possible to see that the performance of every algorithm degrades with the increase of the number of documents. In HBLCoClust this degradation could be perceived only with 10 classes.

Finally, it should be reinforced here that HBLCoClust did not make use of any feature selection algorithm with prior knowledge. On the other hand, the results degradation is noticeable when using an unsupervised selection (PAM) with \(\chi\text{-Sim}\_p\).

In these experiments the coverage of documents clustered by HBLCoClust ranged from 97\% to 100\% causing little

<table>
<thead>
<tr>
<th>Name</th>
<th>M2</th>
<th>M5</th>
<th>M10</th>
<th>NG1</th>
<th>NG2</th>
<th>NG3</th>
</tr>
</thead>
<tbody>
<tr>
<td>#classes</td>
<td>2</td>
<td>5</td>
<td>10</td>
<td>2</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>#docs</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>400</td>
<td>1000</td>
<td>1600</td>
</tr>
</tbody>
</table>

Table I

THE 6 DIFFERENT SUBSETS EXTRACTED FROM THE NG20 DATASET FOR EVALUATION PURPOSE.
Table II
MICRO-AVERAGED PRECISION ALONG WITH STANDARD DEVIATION FOR THE 6 DATASETS FOR EACH ALGORITHM.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>M2</th>
<th>M5</th>
<th>M10</th>
<th>NG1</th>
<th>NG2</th>
<th>NG3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSA</td>
<td>0.92 ± 0.02</td>
<td>0.87 ± 0.06</td>
<td>0.59 ± 0.07</td>
<td>0.96 ± 0.01</td>
<td>0.82 ± 0.03</td>
<td>0.74 ± 0.03</td>
</tr>
<tr>
<td>ITCC</td>
<td>0.79 ± 0.06</td>
<td>0.49 ± 0.10</td>
<td>0.29 ± 0.02</td>
<td>0.69 ± 0.09</td>
<td>0.63 ± 0.06</td>
<td>0.59 ± 0.05</td>
</tr>
<tr>
<td>BVD</td>
<td>0.95 ± 0.00</td>
<td>0.97 ± 0.01</td>
<td>0.80 ± 0.04</td>
<td>0.98 ± 0.00</td>
<td>0.94 ± 0.01</td>
<td>0.90 ± 0.02</td>
</tr>
<tr>
<td>(\chi^2)-Sim (SMI)</td>
<td>0.81 ± 0.10</td>
<td>0.79 ± 0.05</td>
<td>0.55 ± 0.04</td>
<td>0.81 ± 0.02</td>
<td>0.72 ± 0.02</td>
<td>0.64 ± 0.04</td>
</tr>
<tr>
<td>(\chi^2)-Sim (PAM)</td>
<td>0.95 ± 0.02</td>
<td>0.96 ± 0.03</td>
<td>0.88 ± 0.07</td>
<td>0.98 ± 0.01</td>
<td>0.97 ± 0.01</td>
<td>0.95 ± 0.03</td>
</tr>
</tbody>
</table>

Figure 1. HBLCoClust running time regarding the size of the dataset.

impact on real-world applications. Regarding the scalability of HBLCoClust, it was performed an experiment using the same parameters as used on NG3 dataset with different sized data. The linear complexity can be seen in Fig. III.

IV. CONCLUSION

In this paper it was proposed a novel Co-Clustering algorithm, named HBLCoClust, capable of finding co-clusters in text data.

This algorithm has the advantage of allowing overlapping among co-clusters, making it easier to detect documents that belong to more than one topic, and also allowing documents or features to be clustered together even if they are not entirely similar. Also the complexity of this algorithm is linear in time and, depending of the implementation, in memory space as well, making it scalable and applicable to large datasets.

The experiments showed that HBLCoClust could find document clusters with higher micro-precision than five other approaches while maintaining a low processing time. The most noticeable of these results is that HBLCoClust improved the precision without the need of a supervised feature selection approach, differently from the other algorithms, which represents a scenario closer to a real-world application.

ACKNOWLEDGMENT

The author would like to thank UFABC for its support and Hussain, S.F. for the experimental datasets.

REFERENCES


