A High-dimensional Access Method for Approximated Similarity Search in Text Mining

F. Artigas-Fuentes, R. Gil-García
CERPAMID. Universidad de Oriente. Cuba
{artigas,gil}@cerpamid.co.cu

J.M. Badía-Contelles
Dpt. ICC, Univ. Jaume I. Castellón. España
badia@icc.uji.es

Abstract

In this paper, a new access method for very high-dimensional data space is proposed. The method uses a graph structure and pivots for indexing objects, such as documents in text mining. It also applies a simple search algorithm that uses distance or similarity based functions in order to obtain the k-nearest neighbors for novel query objects. This method shows a good selectivity over very-high dimensional data spaces, and a better performance than other state-of-the-art methods. Although it is a probabilistic method, it shows a low error rate. The method is evaluated on data sets from the well-known collection Reuters corpus version 1 (RCV1-v2) and dealing with thousands of dimensions.

1. Introduction

Searching the most similar documents to a given one is crucial in text mining because it is the basic process of many techniques such as classification or information retrieval. Two of the major issues that text mining faces are the large amount of documents, millions in modest cases, and a very high dimensionality of the featured space. Text documents are usually represented as vectors, where each dimension corresponds to a term and the value reflects its importance in the document.

There are many approaches to find the exact vicinity of an object. However, they suffer the curse of dimensionality, that is, their performance drastically decreases as the number of dimensions grows. This problem prevents its application in text mining.

To avoid the curse of dimensionality a variety of methods based on inexact searching have been proposed. In [1, 2] a probabilistic technique, with a good performance, was presented. This solution uses some elements of the training set as pivots or permutants. Basically, the permutants are used to predict proximity between elements and to reduce the number of real distance evaluation at query time. Although this method has a good performance when searching proximities over documents, it introduces an overload at search time, due to the necessity to perform a sequential search over permutants [1], or to use an auxiliary structure to avoid it [2]. This overload increases when the space dimension or the size of datasets grows.

In this paper we introduce improvements to our access method for indexing collections of objects representing a very high-dimensional space presented in [3]. For indexing, this method uses a combination of a graph structure and pivots (used as entry points), and a very fast search algorithm that uses distance or similarity based measures in order to obtain the k-nearest neighbors (k-nn) of novel query objects. In this paper, we introduce a new fast way to generate the connected graph and a prune rule to improve searches.

Although the time required to generate the index structure grows with the size of collection of objects used, this process is carried out only once (offline) and does not affect the query process.

The rest of the paper is organized as follows: Section 2 describes the access method. The experimental results are discussed in Section 3. Finally, we expose our conclusions in Section 4.

2. Access method

In this section, the access method presented in [3] is briefly explained, and the improvements are introduced. The method involves two main phases. In the first one, the graph structure is constructed using a database of objects. In the second phase, the neighborhood of novel objects is found.

2.1. Index structure generation

The idea of our index structure is to create a connected and non-oriented graph where each element is

1051-4651/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPR.2010.772

joined with its θ nearest neighbors. This idea was first presented by Dickerson in 1996 [4], using a variant of Delaunay triangulation, but this method is not always applicable to documents because it requires more objects than dimensions. The Dickerson method permits to obtain the exact vicinity of already indexed objects, not of novel ones. To solve that problem, we choose, using a certain criteria, a few number of indexed elements as entry points to the structure.

In [3] we present a brute force method to construct the graph index structure, but the time required to generate it grows fast with the size of dataset. In this paper, we present a new and more efficient algorithm to perform this task. But first, we describe two conditions used in some steps of our method:

- C1: $u_k \in U$ is external to $O \subset U$, for the relationship Ψ, iif $\Psi(u_k, \overline{u}) < \Psi(o_i, \overline{u})\ \forall o_i \in O$.
- C2: $u_k \in U$ is internal to $O \subset U$, for the relationship Ψ, iif $\Psi(u_k, \overline{u}) > \Psi(o_i, \overline{u})\ \forall o_i \in O$.

were U is the database, $\overline{u}$ is the global centroid of U, O is the set of objects connected with u_k by the graph, and Ψ is the particular distance, similarity or dissimilarity measure used in the specific problem.

The new algorithm is the following:

1. A connected graph is generated using all database objects.
    (a) initially, the objects of the database are sorted by its relationship with the global centroid, and divided into a list L of equal sized subsets.
    (b) let l be the first subset in L.
    (c) the closest pair (o_r, o_t) of objects in l are connected and its centroid $c_{r,t}$ is calculated and saved.
    (d) the closest pair formed by a non-connected object o_e, from l, and a saved centroid $c_{p,q}$ is calculated. Object o_e is connected with the two objects related with $c_{p,q}$ (o_p, o_q), and two new centroids, $c_{e,p}$ and $c_{e,q}$, are calculated and saved.
    (e) step (d) is repeated until all objects of l are connected into the graph.
    (f) set l as the next subset of L, if exists.
    (g) if $l \neq \emptyset$ go to step (d), else exit.
2. The graph is completed by adding edges in such a way that each object is connected with its θ nearest neighbors (calculated using an exhaustive way).
3. The entry-points set is created with objects that fulfill conditions C1 or C2.

The main idea of the new algorithm is to reduce the number of objects to be compared with centroids in each iteration (step (d)). The connected graph obtained is not the same that with the original algorithm, but this is not important because the graph will be completed in step 2.

2.2 Searching

In this phase, novel objects are used as queries. The goal is to obtain the k most related indexed objects to each of these queries. We perform this task in two main steps. If $k = 1$, it is obtained by performing only the step 1 of the algorithm described below. On the contrary, if $k > 1$, the second step of the algorithm is required.

The idea of the searching algorithm is the following:

1. First, the NN object of query q is calculated:
    (a) From the entry-points set, the EP most related objects with q are selected. Those objects are the initial NN solution set.
    (b) From indexed objects connected with any object in the current NN, the most related with q is selected and becomes the new NN solution.
    (c) Step (b) is repeated until no object connected to the current NN solution is nearer to q than it.
    (d) Set kNN, the original solution set, equal to NN solution.
2. If more than one object is needed in the solution:
    (a) Set a search radio equal to the worst possible relationship between two objects as it is defined by Ψ.
    (b) From indexed objects, and connected with any object in kNN solution set, select at most k new objects with relationship value equal or greater than the radio.
    (c) From the union of objects in current kNN solution and objects obtained in previous step, select the k objects most related to q. These objects become the new kNN solution set. If no new objects were found, return kNN and end the algorithm.

(d) Update the radio value with the worst relationship value between objects in the current kNN solution and q.

(e) Go to step (c).

2.3 Prune rule

In order to avoid computing a large number of relationships between objects, a prune rule was applied. When we need to obtain the most related objects to the query (in steps 2.a and 2.b), we first sort the candidate vectors by its decreasing number of coincident dimensions greater than zero with the query. Then, we just compare the query with a fraction of vectors with highest coincidences.

3 Experimental results

In order to compare the new graph method (`nGraph`), the old one (`oGraph`) [3], an exhaustive knn searcher, and two variants [1, 2] of the permutation based indexing method, `Permut` and `Permut+LC`, were implemented, always using the same relationship function.

The Ψ used (1) was a distance function based on the well-known cosine similarity, because it is the most wide used to compare documents in text mining:

$$\Psi(o_1, o_2) = \sqrt{1 - cos(o_1, o_2)} \qquad (1)$$

All the algorithms were implemented using Python 2.5 over an Intel(R) Core(TM)2 Quad CPU, 2.50 GHz and 3GB of RAM with Linux Mandriva 2009 OS.

To perform our experiments, we use the well-known benchmark collection Reuters corpus version 1 (RCV1-v2) [5]. This collection has a set of documents represented as vectors. The feature vector for each document was produced from the concatenation of text in the <headline> and <text> XML elements. Text was reduced to lower case characters, after which tokenization, punctuation removal and stemming, stop word removal, term weighting, feature selection, and length normalization was applied. The LYRL2004 partition, with 23149 training, and 781265 testing vectors, was used. The representation space of this partition has a of 47152 dimensions. Both training and testing subsets were used as database and query objects, respectively.

First, to calculate the impact of our improvement in the indexing phase, different sized indexing structures with all mentioned methods were generated. For that, various vectors of the database were randomly selected. In order to generate graphs, θ=30, and two different subset sizes for step 1.a (s=20 and s=50), were used. For

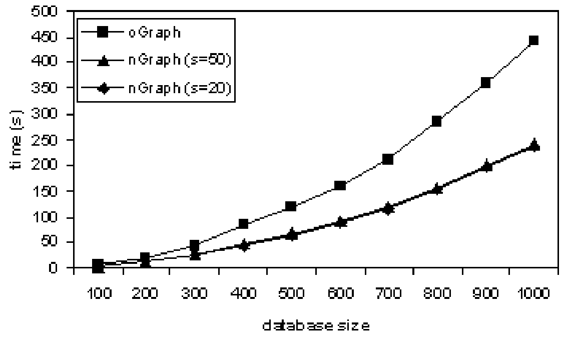

Figure 1. Generation time for different sized training sets. oGraph and nGraph.

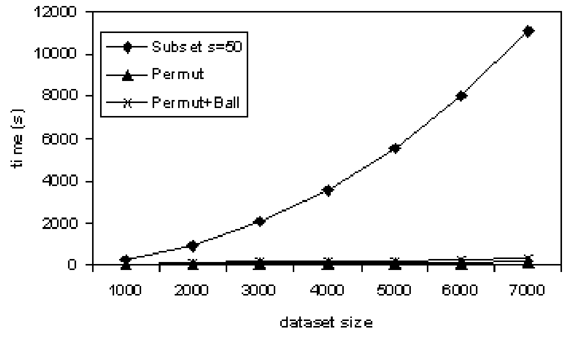

Figure 2. Generation time for different sized training sets. nGraph and permutants.

permutations we use the same parameter values mentioned in its original papers. We use as secondary indexing structure for permutations, the LC exact method reported by E.Chavez et al. in [6]. The processing time comparison between our original and improved algorithms was reported in Figure 1. The comparison with the rest of methods was reported in Figure 2.

We can see in Figure 1 that nGraph reduces the time required by oGraph to generate the graph index structure. Besides, the performance is not impacted by small variations in the subsets size (step 1.a).

On the other hand, Figure 2 shows that the required time for indexing of the others methods is less than ours, but it has not any impact over the search phase.

Next, in order to evaluate the effect of our prune rule in the search phase, the 5-NN of 500 randomly selected testing vectors were computed. For each method, the average search time, the number of right solutions (against the sequential one), and the average percent of relationships computed, were reported in Figures 3 to

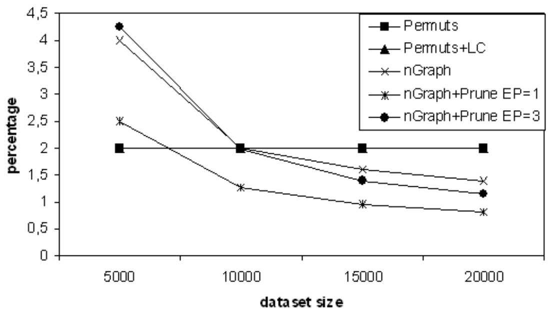

Figure 3. Comparison of average percent of relationships computed.

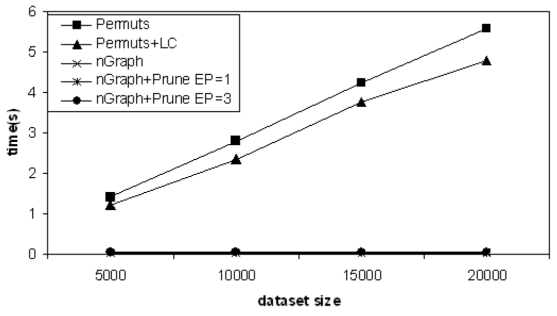

Figure 4. Average search time for each method varying the size of database.

5, respectively. We adjust our prune rule to compare only the 50 percentage of candidate objects. Finally, we select 1 and 3 as values of the EP parameter.

Figure 3 shows that the average relationships computed by our methods decreases as the dataset size increases. On the other hand, Figure 4 shows that the average search time of our algorithm is less than the others for all dataset sizes reported. This happens because we only compute a small number of relationships due to the impact of combination of our data structure and prune rule. For `Permut`, the search cost is $O(k + fn)$, for real objects, plus $O(kn + fn\log n)$, for permutants. Using LC, in the best case, the second cost is reduced to $O(kfn)$ (see [2] for details).

Finally, Figure 5 shows that the number of exact answers (objects) appearing in solution sets for the combination of `nGraph`, prune rule and $EP = 3$, is greater than other variants for all dataset sizes reported.

4 Conclusions

In this paper, an improvement of our previous access method based in graphs is proposed. This new method deals successfully with the curse of dimensionality and can be applied, for example, to text documents with thousands of dimensions. We compare its performance against two state-of-the-arts methods based on permutants. The experimental results show that our method reduces both the average search time and average relationships computed, with similar or even better quality results for dataset sizes reported.

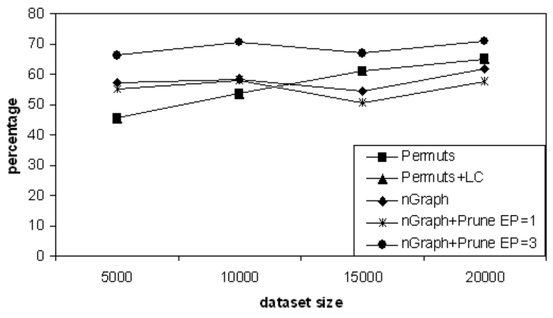

Figure 5. Number of objects in solutions reported by exhaustive search.

References

[1] E. Chávez, K. Figueroa and G. Navarro. Effective proximity retrieval by ordering permutation. TPAMI'07. IEEE Transactions on Pattern Analisys and Machine Intelligence:30(9):1647–1658. Sep. 2008.

[2] K. Figueroa and K. Fredriksson. Speeding up permutation based indexing with indexing. SISAP'09. IEEE Computer Society:107–114. 2009.

[3] F. Artigas-Fuentes, R. Gil-García, J.M. Badía-Contelles and A. Pons-Porrata. Vicinity calculation with graph in text mining. *UCT*,48, F. Genolet (Eds.): 1-10, July 2008.

[4] M. Dickerson. Algorithms for proximity problems in higher dimensions. *Computational Geometry Theory and Applications*, 5:277-291, 1996.

[5] D.L. Lewis, Y. Yang, T.G. Rose and F. Li. RCV1: A new benchmark collection for text categorization research.*Journal of Machine Learning Research*, 5:361-397, 2004.

[6] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363-1376, 2005.