Fast k -NN Classifier for Documents Based on a Graph Structure

Fernando José Artigas-Fuentes¹, Reynaldo Gil-García¹,
José Manuel Badía-Contelles², and Aurora Pons-Porrata¹

¹ Center of Pattern Recognition and Data Mining
Universidad de Oriente, Santiago de Cuba, Cuba
{[@csd.uo.edu.cu](mailto:artigas,gil,aurora)}

² Computer Science and Engineering Department
Universitat Jaume I, Castelló, Spain
badia@uji.es

Abstract. In this paper, a fast k nearest neighbors (k -NN) classifier for documents is presented. Documents are usually represented in a high-dimensional feature space, where their terms are treated as features and the weight of each term reflects its importance in the document. There are many approaches to find the vicinity of an object, but their performance drastically decreases as the number of dimensions grows. This problem prevents its application for documents. The proposed method is based on a graph index structure with a fast search algorithm. Its high selectivity permits to obtain a similar classification quality than the exhaustive classifier, with a few number of computed distances. Our experimental results show that our method can be applied to problems of very high dimensionality, such as Text Mining.

Keywords: nearest neighbor classifier, fast nearest neighbor search, text documents.

1 Introduction

Text classification is the task of assigning documents to one or more predefined classes. This task relies on the availability of an initial set of text documents classified under these classes (known as training data). Classification falls at the crossroads of information retrieval, pattern recognition and data mining, that involves very large data sets. Moreover, the dimensionality of the text documents is usually large. Therefore, it is crucial to design algorithms which scale well with the dimension.

The k nearest neighbor (k -NN) classifier is a very simple and popular approach used in classification [1], but it has the problem of the exhaustive computation of distances to training objects. Several methods have been proposed in order to avoid this problem. One approach involves improving the access methods combining appropriate index structures, such as trees or graphs, with fast search algorithms. But, in the most of cases, their performance drastically decrease

as the number of dimensions grows. This problem is known as “the curse of dimensionality” [2], and prevents its application for text documents.

Several exact methods have been proposed for objects with high dimensionality, such as VA-File [2], VA+-File [3], IQ-tree [4] and more recently VQ-index [5]. The main purpose of those methods is to overcome the I/O disk bottleneck, which is crucial in large databases. Those methods were tested on relative high dimensional spaces, with 32, 64, 200 and 500 dimensions. However the main purpose of our proposal is to work over spaces with several thousands of dimensions, as arise in the case of text documents. In this case we can deal with more than 30.000 dimensions with a relatively small set of text documents.

On the other hand, several fast search algorithms have been proposed by Arya et. al in [6], and others, such the optimal nearest neighbor algorithm for data structures that are stored in main memory.

However, for very high dimensionality most of those methods have a performance as bad as a linear scan, or even worse. Recall that a linear scan does not scale well when the set of objects to search is large or the relationship function (distance, similarity or dissimilarity function) is hard to compute.

Different relaxations on the precision of the result have been proposed in order to obtain a computationally feasible solution in those cases. This kind of methods perform an inexact proximity searching, as opposed to the classical exact proximity searching. Inexact proximity searching is possible in many applications because the preprocessing of data already involves an approximation to reality, and therefore a second approximation at search time is acceptable. Examples of those methods can be found in [7,8].

In [9] an approximated classifier for mixed data was presented. This method uses a tree index structure and a fast search algorithm. It obtains the information necessary to classify while searching. Even when the accuracy of the classification obtained for mixed data is high, it decreases when applied to documents.

In this paper, we introduce a fast k -NN classifier for text documents based on a graph index structure with an approximate k nearest neighbors fast search algorithm. Its high selectivity and precision permits to obtain a similar classification quality than the exhaustive classifier, with a few number of computed relationships. Our experimental results show that it is feasible the use of the proposed method in problems of very high dimensionality, such as Text Mining.

The rest of the paper is organized as follows: Our proposed classifier is presented in Section 2. The obtained experimental results over Reuters Corpus Version 1 (RCV1-v2) are presented in Section 3. Finally, in Section 4, we present some conclusions and future work.

2 Proposed Classifier

In this section, an approximate fast k -NN classifier for documents is introduced. The classifier consists of two phases. In the first phase, the graph structure, using training set T , is constructed. In the second phase, novel documents are classified.

2.1 Preprocessing Phase

The main idea of this phase is to build an index structure based on a connected graph. This graph must fulfil the following conditions:

- Each vertex corresponds to a different and unique training document (represented by its features vector).
- Each edge represents a relationship value between two vertices calculated by a similarity, dissimilarity or distance function Ψ .
- Each vertex v has, at least, ϕ adjacent vertices, where ϕ is an integer value preset by the user. Those vertices correspond with the ϕ nearest neighbors of v .
- Vertices are connected forming triangles of minimum area.
- A small number of vertices are fixed and used as possible entry points to the structure during the search phase.

In [10] the algorithm to build the graph index structure was introduced, whereas in [11] some improvements in order to reduce the time cost to build it were presented.

It is important to describe how the triangles of the graph are built. For this access method, there are two ways to perform this task.

The simple way is to connect first the most Ψ -related pair of documents in the training set TS . This pair of documents are used to obtain the first two vertices and the first edge of the graph. Then, the rest of documents in TS are candidates to be connected to this pair in order to obtain the third vertex of the first triangle. To do that we calculate the *media object* of this pair of vertices. It is obtained by adding each dimension of original objects and inserted into the set of media objects MOS . Then, the most Ψ -related pair formed by a candidate and a member of MOS is computed, and the candidate of this pair is selected as the next vertex. Following a similar iterative process new media objects for the new edges are calculated and inserted into MOS , and the rest of documents become candidates to be connected into the graph.

The second method to build the triangles reduces the number of candidates to be considered in each iteration to obtain the new vertex to be connected into the graph. This improvement also reduces the total time to generate the index structure. In this case, the documents in TS are previously sorted by their relationship with the global centroid GC . This centroid is calculated in the same way of media objects, but using all documents in TS .

Then, the sorted set of documents is divided into a number of equal sized subsets. As result we obtain a sequence of subsets ordered by its decreasing relationship with GC . Next, the first subset is selected as the current one (CS). In this way, on each iteration, we only consider as candidates to become new vertices those belonging to CS . When all candidates have been connected, the next subset becomes CS , and the next iteration begins. This process continues until all documents are connected into the graph.

To guaranty the ϕ condition, if any vertex results with less than n of adjacent vertices, that vertex is connected with its other $\phi - n$ nearest neighbors. The ϕ nearest neighbors of all documents are calculated and kept in a previous stage.

Finally, vertices belonging to the borders of the region defined by the graph are selected as entry points to the structure (See [10,11] for details). Those vertices are called as Entry Points Set (EPS). This is the main difference of this access method: the index structure has several entry points, while methods based on trees have only one entry point to the structure, the root.

2.2 Classification Phase

In this phase, given the index structure G previously built, the classes associated with the documents in the training set, and a novel document d to be classified, our classifier finds its k nearest neighbors, according with Ψ , and assigns to d its majority class.

The classification involves three main stages:

1. The k nearest neighbors of the novel document are searched.
2. The votes of classes are counted, using a vote rule.
3. Finally, the classification is performed using a decision rule.

In the first main stage, the fast search algorithm proposed in [10] and improved in [11] is used. It has three steps, in the first one a proper entry point EP_d to G for the current search d is selected. This choice can be different for each d and it must be the most Ψ -related to d of the members of EPS . EP_d is calculate in an exhaustive way, and it becomes the current solution (NN).

In the second step of the search the current nearest neighbor vertex to d is found. This task is performed by traversing the index structure following the edges of graph. The next solution is the most Ψ -related adjacent vertex of NN to d , if it is better than the current one. The process ends when there is no new NN . The problem is that it is not always possible to obtain the nearest neighbor. In a few cases an approximated one must be obtained.

In order to improve the quality of k -NN, a variation of the search algorithm and a prune rule were introduced in [11]. The variation involves selecting the best results of three independent searches of the k -NN, using three different entry points to the index structure. This solution increases the number of comparisons computed during searches. To avoid this problem the pruning rule that increases the selectivity of the search algorithm, avoiding extra computations was used.

In the last step of the search, if $k > 1$, the other $k - 1$ neighbors of d are obtained. This task is performed using another algorithm described in [10]. It also traverses the edges of the graph, but using as initial point the actual (or approximated) NN calculated in the previous step. Besides, the algorithm keeps in each iteration a vector with the current list of k nearest neighbors.

After finding k -NN, the votes of each class are counted and the majority class is assigned to d .

3 Experimental Results

In this section, the results of applying the proposed fast approximate k -NN classifier are presented.

To perform our experiments, we use the well-known benchmark collection Reuters Corpus Version 1 (RCV1-v2) [12]. 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 23.149 training, and 781.265 testing vectors, was used.

Classes files of both training and test sets were modified to avoid overlapping among classes. The resulting sets belong only to four non-overlapped classes: ECAT, CCAT, MCAT and GCAT. This modification was necessary because the other fast approximated classifier implemented (FC) to compare with our proposal (FGC) do not support class overlapping. A k -NN exhaustive classifier (EC) was implemented too, and was used as base line of the comparison.

First, a 10% of training documents (692) documents were randomly chosen to build index structures for both FC and FGC classifiers, while maintaining the distribution of the class probabilities in the original training and test sets. The representation space obtained has 8.731 dimensions.

FC is a k -NN classifier [9] that uses an index structure based on a tree. Each node of the tree contains a certain number of elements selected using a grouping algorithm. In the original paper, the authors present and use a new clustering algorithm called *CMFS*, which is an extension of the C-Means algorithm [13], but in our experiments we used both *CMFS* and the well-known *K-Means* [14].

Besides, FC requires additional parameters to build the tree. The minimal number of objects in a node was fixed to 20, the maximum number of clusters by level was fixed to 5, and the maximum number of iterations of the clustering algorithm was fixed to 10. The authors used three stop conditions to determine the leaf nodes. But, we use only the last two conditions based on non-homogeneous (noHomo) and homogeneous (Homo) nodes. In the case of our proposal, the value of ϕ was fixed to 50.

We select a small number of training documents due to the cost of *CMFS* when applied to grouping objects with an elevate number of dimensions.

The Ψ used was a distance function based on the well-known cosine similarity ($\Psi(d_1, d_2) = \sqrt{1 - \cos(d_1, d_2)}$), where d_1 and d_2 are the documents to be compared, which is the most widely used to compare documents in text mining. This measurement reaches its minimum value at 0 and maximum at 1. All the classifiers use the same vote rule.

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.

For the classification phase, 500 documents from the test set were randomly selected. The classification was carried out using the three classifiers: FC, FGC and EC. For FC, its authors offer two algorithms to search the approximates k -NN. But, we only show the results obtained using the KMSNLocal algorithm, because the results for the other were very similar.

Fig. 1. A comparison of the quality of the implemented classifiers varying the number of nearest neighbors calculated

To compare the quality of classifiers, macro F1-measure was used. It is the average on the F1 scores of all the topics. The F1 score ($F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$) can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and its worst score at 0. Precision is the proportion of documents classified into a class that indeed belong to it, and recall is the proportion of documents belonging to a class that are indeed classified into this class. When quality is computed for several categories, the results for individual categories must be averaged in some way. Two methods may be adopted: micro-averaging and macro-averaging. Micro-averaging gives equal weight to every document, while macro-averaging gives equal weight to each category.

Figure 1 shows the macro-average F1 values obtained with the classifiers varying the number of nearest neighbors computed from 1 to 5. It can be seen that our classifier was better than FC in all cases. Surprisingly, our proposal even improves the EC classifier, although this uses an exact method to obtain the vicinity of document queries.

In order to investigate the cause of this behavior, we implemented a fourth classifier β FGC, based on FGC. Despite of the original, in the voting phase, for β FGC only those nearest neighbors with a relationship value greater than a certain threshold β are taken into account. We do the same change to EC and obtain a new base line classifier (β EC).

Figure 2 shows the quality of results when β has the values 0.75, 0.80, 0.85 and 0.90. For β values less than 0.90 the quality of β FGC are slightly worse than those of β EC. On the contrary, for β values greater than 0.90 it is slightly better. This results means that the previous strange behavior was provoked by the elements of the k -NN more distant to the queries. When all k -NN are taken into account, in most of the approximate results, the last elements found by our search algorithm belongs to the actual classes of queries despite of the elements found by EC.

Fig. 2. A comparison between β EC and β FGC varying β parameter

Another aspect that we consider to evaluate our proposal, was the time required to classify documents. The best improvement obtained as consequence to use our access method based on a graph was the dramatical reduction of the number of distances computed to obtain the k -NN against the 100% needed by the EC. Table 1 shows the time cost and the best quality in classification obtained by EC, FC and our proposal FGC.

Table 1. Summary of best results for each classifier

Parameter	EC	FC	FGC
Time (s)	60.06	1.20	14.53
Macro-average F1	0.71	0.45	0.73

As you can see, FC is very fast, but its results are not good for documents classification. On the other hand, our proposal obtains good results in a reasonable time.

4 Conclusions

In this work, an approximated fast k -NN classifier for text documents, a problem with a very high dimensionality, was proposed. In order to compare our method, different variants of a fast k -NN classifier were implemented using the same distance function to compare documents, and the same vote and decision rules for classification. An exhaustive classifier was used as base line. Experimental results show that our proposal obtains high quality results, better than the state-of-the-art classifiers presented.

Our method reduces drastically the time of the exhaustive classifier, while obtaining very similar or even better classification quality results. This behaviour is due to the use of an index structure based on a connected graph and a fast search algorithm.

References

1. Myles, J.P., Hand, D.J.: The Multi-Class Metric Problem in Nearest Neighbor Discrimination Rule. *Pattern Recognition* 23, 1291–1297 (1990)
2. Schek, H., et al.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: *VLDB 1998*, New York, USA, pp. 194–205 (1998)
3. Ferhatosmanoglu, H., et al.: High dimensional nearest neighbor searching. *Information Systems* 31, 512–540 (2006)
4. Berchtold, S., et al.: Independent quantization: An index compression technique for high-dimensional data spaces. In: *Proc. 16th Int. Conf. on Data Engineering*, San Diego, CA, pp. 577–588 (2000)
5. Tuncel, E., et al.: VQ-Index: An Index Structure for Similarity Searching in Multimedia Databases. In: *10th ACM International Conf. on Multimedia 2002*, Juan Les Pins, France, pp. 543–552 (2002)
6. Arya, S., et al.: An optimal algorithm for approximate nearest neighbor searching. In: *5th Ann. ACM-SIAM Symposium on Discrete Algorithms*, pp. 573–582 (1994)
7. Chávez, E., et al.: Effective proximity retrieval by ordering permutation. In: *IEEE Trans. on Pattern Analysis and Machine Intelligence*, TPAMI 2007, vol. 30(9), pp. 1647–1658 (2008)
8. Figueroa, K., Fredriksson, K.: Speeding up permutation based indexing with indexing. In: *SISAP 2009*, pp. 107–114. IEEE Computer Society, Los Alamitos (2009)
9. Hernández-Rodríguez, S., et al.: Fast Most Similar Neighbor Classifier for Mixed Data Based on a Tree Structure. In: Rueda, L., Mery, D., Kittl, J. (eds.) *CIARP 2007*. LNCS, vol. 4756, pp. 407–416. Springer, Heidelberg (2007)
10. Artigas-Fuentes, F., et al.: Vicinity calculation with graph in text mining. In: *Genolet, F. (ed.) UCT*, vol. 48, pp. 1–10 (2008)
11. Artigas-Fuentes, F., et al.: A High-dimensional Access Method for Approximated Similarity Search in Text Mining. In: *ICPR 2010 Congress*, Istanbul, Turkey (2010)
12. Lewis, D.L., et al.: RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research* 5, 361–397 (2004)
13. Yu, J.: General C-Means Clustering Model. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 27(8), 1197–1211 (2005)
14. MacQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. In: *Proc. 5th Berkeley Symp. Math. Statistics and Probability*, vol. 1, pp. 281–297 (1967)