Indexing Support Vector Machines for Efficient top-k Classification

Giuseppe Amato, Paolo Bolettieri, Fabrizio Falchi, Fausto Rabitti, Pasquale Savino
ISTI-CNR
Pisa, Italy
Email: g.amato@isti.cnr.it, p.bolettieri@isti.cnr.it, f.falchi@isti.cnr.it, f.rabitti@isti.cnr.it, p.savino@isti.cnr.it

***Abstract*—This paper proposes an approach to efficiently execute approximate *top-k classification* (that is, identifying the best k elements of a class) using Support Vector Machines, in web-scale datasets, without significant loss of effectiveness. The novelty of the proposed approach, with respect to other approaches in literature, is that it allows speeding-up several classifiers, each one defined with different kernels and kernel parameters, by using one single index.**

***Keywords*-Image Classification; Support Vector Machines; Similarity Searching.**

I. Introduction

The classification problem is typically defined as follows. Given a set of classes $c_1, \ldots, c_n$, and an object o, the classification problem is to decide which classes o belongs to. In this paper, on the other hand, we address the classification problem from an *Information Retrieval* perspective. Let $c_1, \ldots, c_n$ be n classes, and DS a very large dataset (for instance the World Wide Web) of unclassified objects. Given a class c_i, we want to retrieve the k objects of DS having the highest likelihood to belong to c_i. In this case, the class c_i can be considered as a query, and the best k objects that belong to c_i as the answer to the query. We call *top-k classification* this formulation of the classification problem.

Support Vector Machines (SVMs) [1], are widely used to perform automatic supervised classification. The aim of this paper is to propose a strategy that, given a set of SVM classifiers defined for a set of classes $c_1, \ldots, c_n$, and a dataset DS, executes top-k classification very efficiently. More specifically, we do not address the problem of learning classifiers, for which several solutions already exist as mentioned in Section III. Rather, given an SVM classifier for any class c_i, and a dataset DS, our aim is to efficiently search in DS for the best k objects that belong to c_i.

As we will see, we propose an approximated approach. That is, our approach returns an imprecise result, compared to the result that would have been obtained by performing a sequential scan of the entire dataset DS and applying the available classifiers to every object. However, the experiments will show a small imprecision, compared to an improvement of efficiency of orders of magnitude.

The rest of the paper is organized as follows. First we discuss related work. Next we briefly introduce the SVM. In Sections IV and V, we present the top-k classification, while in Section VI, the index structure used in the experiments. Finally in Sections VII and VIII, we describe the settings of the experiments and the analysis of the results.

II. Related Work

Efficient top-k classification techniques were proposed in [2], [3], by leveraging on the property that instances in the feature space lie on a hypersphere. This approach is able to use one index for various classes obtained using Support Vector Machines and built using the same kernel. In [4], the authors propose a method for efficient top-k classification based on boosting. In [5], the authors propose an efficient method for retrieving the instances closest to the separating hyperplane (the most ambiguous instances) to support active relevance feedback.

The novelty of the proposed approach, with respect to other methods existing in literature, is that it allows supporting and speeding-up the use of several classifiers, each one defined with different kernels and/or kernel parameters, by using one single index in the *input space* [1] of the dataset.

III. Introduction to SVM

An SVM [1] builds classifiers by learning from a training set that is composed of both positive and negative examples.

In many cases, in order to be able to separate element that belong to the class from those that do not belong to the class, it is convenient to map vectors, representing elements, in an higher dimensional vector space using a mapping function $\Phi(\cdot)$. Omitting several theoretical details (see [1] for more information), the learning phase determines a vector ω such that the decision function

$$f(o) = < \omega, \Phi(o) > + b \tag{1}$$

is able to optimally classify most of the training set examples ($< \omega, \Phi(o) >$ is the dot product between vectors ω and $\Phi(o)$). When the decision function is positive it indicates that an object belongs to the class. A popular learning algorithm is the kernel-based version of the adatron algorithm.

The SVM literature often call *input space* the space where objects are defined, and *feature space* the space where

[1] the space in which objects are originally represented (see Section III).

 ISBN: 978-1-61208-129-8

objects are mapped by $\Phi(\cdot)$. We will use this terminology in the reminder of the paper.

SVM methods do not define the mapping function explicitly, but use the properties of the kernel functions to perform learning and classification. A kernel function K, defined as $K(o_i, o_j) =< \Phi(o_i)^T, \Phi(o_j) >$, computes the dot-product of o_i and o_j in the feature space. There are simple kernel functions that easily compute the dot-product of objects mapped in very high or even infinite dimensional spaces without even knowing the actual mapping functions.

It can be proven [1] that the kernel-based decision function defined above, can be also represented in the dual form

$$f(o) = \sum_{(o_i,y_i)\in T} y_i \alpha_i K(o, o_i) + b \qquad (2)$$

where o_i are the element of the training set and y_i is 1 or to -1 according to the fact that the training object o_i is a positive or a negative sample of the class to learn. In this formulation, the learning phase consists in finding the parameters α_i, which basically determine the contribution of each example o_i of the training set to the solution of the learning problem, rather than the vector ω. Most of the α_i, obtained in the training phase, will be equal to 0. So, in order to compute the decision function f, we only need to maintain the training objects for which the α_i are greater than 0. These objects are the *support vectors*.

IV. Top-k classification

Let f_c be a decision function defined according to Equation 2 for the class c. The value $f_c(o)$ indicates the degree of membership of the object o to the class c. Large positive values indicate high membership of o to c; large negative values indicate that o does not belong to c; values close to zero indicate uncertainty.

The top-k classification problem can also be formulated as follows. Given a decision function f_c and a dataset DS, retrieve the k objects $o_1, \ldots, o_k$ in DS for which $f_c(o_i), i = 1 \ldots k$, is larger than when applied to any other object in DS. More formally:

Definition: 1: Let DS be a dataset of objects, c a class, and f_c the decision function for c. We define

$$\begin{aligned} top\text{–}k(DS, c) = &\{o_1, \ldots, o_k \in DS \mid \\ &\forall o \in (DS \backslash \{o_1, \ldots, o_k\}), \\ &f_c(o) \le f_c(o_i), i = 1, \ldots, k\} \end{aligned}$$

A. Approximate top-k classification

Clearly, $top{-}k(DS, c)$ can be computed with a sequential scan of the whole dataset. However, this is very inefficient when DS is very large. Suppose we have a set of candidates $CS \subseteq DS$ for class c. Then, top-$k(CS, c)$ is an approximation of top-$k(DS, c)$. However, consider that if CS is chosen carefully, top-$k(CS, c)$ will not necessarily differ very much from top-$k(DS, c)$. For instance, if $CS =$ top-$k(DS, c)$, then top-$k(CS, c)$ =top-$k(DS, c)$. According to this, given CS, approximate top-k classification can be performed by applying the decision function to the objects of CS rather than all objects in DS. Provided that CS is much smaller that DS ($\#CS \ll \#DS$[2]), this process will be much more efficient than exhaustively classifying all objects in DS.

In the next section, we will propose a strategy to obtain CS, by using techniques of nearest neighbors searching, in such a way that approximation will be highly accurate and CS is much smaller than DS.

V. top-k classification by means of Nearest Neighbors Search

The training set T_c for a class c consists of positive and negative examples. Let us denote as PT_c and NT_c respectively the positive and negative training objects ($T_c = PT_c \bigcup NT_c$). As discussed in Section III, the learning phase identifies the αs parameters for the decision function, and implicitly the support vectors (the training vectors whose α_i are strictly greater than 0). Let us denote as $PSV_c \subseteq PT_c$ and $NSV_c \subseteq NT_c$ respectively the positive and negative support vectors identified after the training of the classifier for a class c. The decision function given by Equation 2 can be rewritten as

$$f_c(o) = \sum_{p\in PSV_c} \alpha_p K(o, p) - \sum_{n\in NSV_c} \alpha_n K(o, n) + b$$

The formula above just uses the support vectors and disregards the elements of the training set whose αs are 0, since they do not provide any contribution to f_c.

From the definition of f_c given above, it is easy to see that the objects o of the dataset that are very similar, according to K, to several positive support vectors and are dissimilar to several negative support vectors, have higher chances to return an high value when f_c is applied to them. In fact, the kernel K can be seen as a similarity function. That is, K returns large values when the two compared objects are similar and small values when the two objects are not similar. This suggests a strategy to select from DS a subset of promising candidates for c. In short, we can first search the objects of the dataset that are closer, according to K to each positive support vector. Then, we apply the decision function f_c only to the selected candidates to find the best k matches.

More formally, the candidate set $CS_c \subseteq DS$ for class c can be obtained as

$$CS_c = \bigcup_{p\in PSV_c} NN_K(p, s, DS) \qquad (3)$$

where $NN_K(p, s, DS)$, is a nearest neighbors query, which returns the s objects of DS most similar to p,

[2] we use $\#$ to indicate the cardinality of a set

according to kernel K, for some s much smaller than the size of DS.

The selection of the k best objects matching the class c can now be obtained by applying f_c only to objects in CS_s, which is significantly smaller than DS.

VI. Using an index structure in the input space

Several scalable techniques can be found in literature to efficiently process nearest neighbors search queries [6], [7]. However, even if, leveraging on these techniques, Equation 3 can be computed very efficiently, there are two reasons why it is not practical to build an index using K directly (that is in the feature space):

1) Access methods for similarity search typically rely on the fact that the similarity (the kernel K in our case) can be expressed in terms of a distance (dissimilarity) function, which should satisfy some specific properties, like for instance the metric postulates. A distance function can always be derived from a kernel K. However, such function, in many cases is not suitable to be used to build an efficient index structure. In fact, given that the kernel K compares elements in an high dimensional feature space, the underlying distribution of distances might not be convenient and we might incur in the curse of dimensionality [8].
2) We would like to support several classifiers for the same dataset, each recognizing a different class. Different classifiers might require different kernels and different kernel parameters (that is, different similarity functions). If we succeed to find a suitable distance function for a certain kernel K and we create an index with this distance, we are bound to the specific kernel K and its kernel parameters (for instance different σs in the case of the RBF kernel), so we are bound to a specific classifier. To support several classifiers, we would need several indexes, each for a specific kernel and kernel parameters.

Next section shows how to solve the above problems by building *one single index in the input space* to serve various kernels, provided that they satisfy some conditions.

A. Kernels that allow using a single index in the input space

Instead of deriving a distance function from a kernel, in many cases it is possible define a kernel in terms of a convenient distance function as follows:

$$K(o_1, o_2) = g(d(o_1, o_2)) \tag{4}$$

where $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$ is a distance function between objects of the input space $\mathcal{D}$.

Many widely used kernels can be defined using Equation 4. For instance, if d is the Euclidean distance (L_2), with $g(x) = e^{-\frac{x^2}{2\sigma^2}}$ we obtain the RBF kernel; with $g = -x^\beta$, for $\beta > 0$, we obtain the power kernel; when d is the L_1 distance, with $g(x) = e^{-\gamma x}$ we obtain the Laplacian kernel. Other examples exist.

Let $\overline{\mathbb{R}} = d(\mathcal{D}, \mathcal{D}), \overline{\mathbb{R}} \subseteq \mathbb{R}$, that is $\overline{\mathbb{R}}$ is the set of possible values that d can give for any arbitrary pair $o_1, o_2 \in \mathcal{D}$. If g is monotonously decreasing over $\overline{\mathbb{R}}$, then the k objects closest to o, with respect to d, are the k objects most similar to o in the feature space, with respect to K.

In other words, *given any kernel K defined according to Equation 4, with a monotonous decreasing g and the same distance d,we have that*[3] $NN_K(p, s, DS) = NN_d(p, s, DS)$.

This implies that the most similar objects to a support vector in the feature space induced by K, are exactly the objects closest to the same support vector, in the input space. Therefore, we can use just one single access method defined using d and built in the input space, to search for the nearest neighbors in the feature space induced by a large class of kernels defined in terms of d. This, as discussed in Section V, also gives us the possibility of identifying the subset of promising candidates just working in the input space, for the same class of kernels[4].

Note also that the gs, that produce the RBF, Laplacian, and power kernels, are all monotonously decreasing in $\mathbb{R}^+$. Therefore, in this case, any d, which always returns positive values, allows this technique to be used.

VII. Experiment settings

Tests of the proposed techniques were executed on a single 2.4GHz Core 2 Quad CPU, using the CoPhIR dataset [11]. CoPhIR consists of 106 millions images, taken from Flickr, described by MPEG-7 visual descriptors. In the tests we used the first set of one millions images taken from CoPhIR. The access method used to efficiently search for objects close to the support vectors, in the input space, is the MI-File [12] (Metric Inverted File). The MI-File is a disk maintained index, based on inverted files, which supports efficient and scalable approximate similarity search on data represented in metric spaces. To define the kernel for the support vector machine, according to Equation 4, we used $g(x) = e^{-\frac{x^2}{2\sigma^2}}$, and as d we used a combination of MPEG-7 distance functions [13]. We trained the support vector machine to recognize 5 different classes: churches, pyramids, seascapes, paintings, and temples. We used a standard kernel-based adatron with cross-validation, to learn

[3] Here we abuse with the notation, NN_K gives the most similar, while NN_d gives the closest ones.

[4] Note that kernels used with SVM must be positive definite [1] or conditionally positive definite [9]. Therefore when a kernel K is obtained from Equation 4, we must first prove this. However, in many common cases this is true. Consider that, [10], in Theorem 12, shows that when $g = e^{-tx}$, for all $t > 0$, K is positive definite iff d is negative definite and symmetric. Note that the Eucledian distance is negative definite and symmetric. In fact, given that the RBF Kernel is positive definite, then d^2, when d is the Eucledian distance, is negative definite and symmetric and, as consequence of Theorem 11 in [10], also d (the Eucledian distance) is negative definite and symmetric.

 ISBN: 978-1-61208-129-8

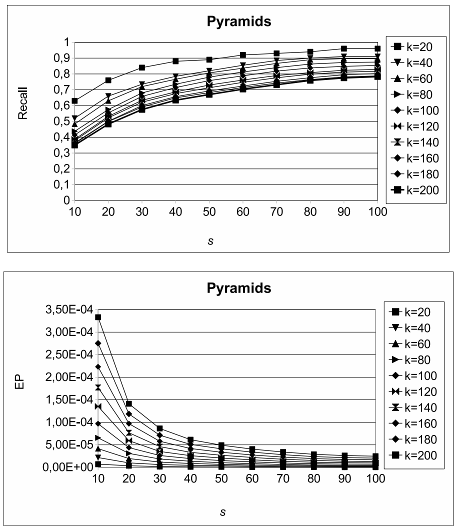

Figure 1. Quality of the of the approximate Top-k classification varying the number of nearest neighbors (s) retrieved for each support vector, according to Equation 3, for various values of k. Here, for brevity, we just show the results when the query is class Pyramids. Similar results were obtained for the other classes.

these classes. The set of candidates CS_c for a class c was obtained according to Equation 3. $NN_d(p, s, DS)$ searches were executed efficiently using the MI-File populated with all objects in the dataset according to distance d. The number s of nearest neighbors to each support vector ranged from 10 to 100, that is respectively 100.000 and 10.000 times smaller than the size of the dataset. Approximate Top-$k(DS, c)$ was executed computing Top-$k(CS_c, c)$ for various values of k ranging from 10 up to 200.

VIII. Analysis of the experiments

It is important to stress that this paper does not propose a new classification technique. Rather, given an SVM classifier built using standard tools, we propose a technique to perform top-k classification much faster than exhaustively classify all objects of a huge dataset. In this respect, given a classifier, our experiments aim at comparing the techniques that we propose, for efficient approximate top-k classification, against the exhaustive solution for top-k classification, which solve the top-k problem by sequentially and systematically classifying all objects of the dataset.

The evaluation of the quality of the top-k classification results consists of two parts: 1) an objective and quantitative evaluation of the error introduced by the use of the approximate classification and 2) a subjective evaluation based on real user feedback.

Both evaluations required to perform an *exhaustive classification* (sequential classification of the entire dataset), that was compared with the proposed approximate technique.

The objective evaluation was carried out by computing the measures of recall and error on the position [6]. More precisely, given a class c, the recall at k, R_k, is the percentage of the best k objects retrieved by the approximate method that also appear in the best k identified by the exhaustive classification as belonging to c.

The error in the position at k (EP_k) measures the quality of the ranking obtained by the approximate method with respect to the exhaustive one. It gives the average shifting of elements in the rank in percentage with respect to the size of the dataset.

More formally, recall at k is

$$R_k = \frac{\#(S_k \cap S_k^A)}{\#S_k} \tag{5}$$

and the error on position at k is

$$EP_k = \frac{\sum_{o \in S_k^A} |OX(o) - S_k^A(o)|}{\#S_k^A \cdot \#X}, \tag{6}$$

where S_k and S_k^A are the k best matches to c found respectively by the exhaustive classification of the entire dataset and by the our approximate method. OX is the ordering of the entire dataset X with respect to the decision function f_c for class c. For example, if o_1 is the most appropriate object that belongs to the class c, o_2 is the second and o_3 is the third, $OX(o_1) = 1$, $OX(o_2) = 2$ and $OX(o_3) = 3$. $S_k^A(o)$ is the position of o in the rank of k best matches found by the approximate classification.

The subjective evaluation, based on user feedbacks was performed by asking 5 students to blindly judge the results obtained with the exhaustive and approximate classification. To compare the two results we used the precision at k measure defined as follows:

$$P_k = \frac{\#(S_k \cap S^c)}{\#S_k} \tag{7}$$

where S_k is the result obtained by either the approximate or the exhaustive classification method, and S^c is the set of images correctly classified for c. $S_k \cap S^c$ was obtained by asking the users to select the correct results in S_k (blindly for exhaustive and approximate classification). Precision at k tells us the percentage of the k retrieved elements that belong to the class c according to the user judgement.

A. Approximate vs exhaustive classification

We first performed experiments to see how, according to Equation 3, the choice of the number s of retrieved nearest neighbor dataset objects to a support vector affects the quality of the approximation. Results are reported in Figure 1. We varied s from 10 to 100. For brevity, in the Figure we report results just for Pyramids. However, similar considerations can be made for the other classes. We can see that, in the chosen range of s, recall increases with s and saturates when s is around 90. On the other hand, the error

 ISBN: 978-1-61208-129-8

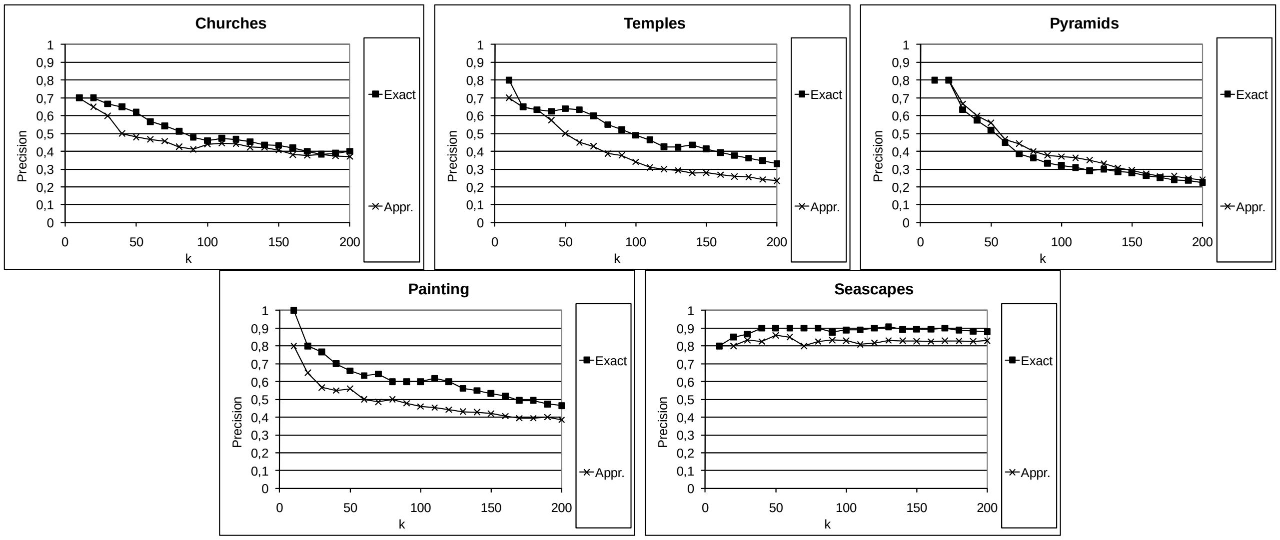

Figure 2. Precision of the approximate Top-k classification and the exact Top-k classification, for various values of k, as judged by users.

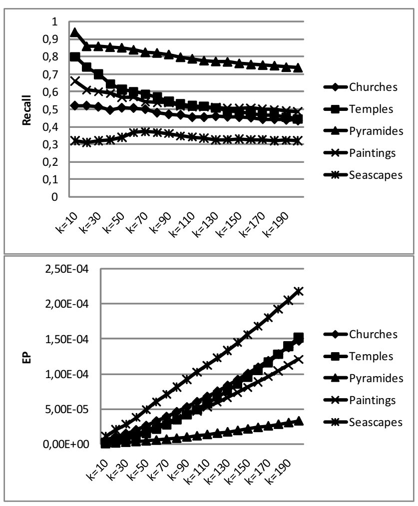

Figure 3. Recall and EP of the approximate top-k classification for several classes as queries, when s is fixed to 100, for various values of k, considering the exact top-k classification the ground truth.

on position decreases rapidly when s increases. Very small error values are already obtained when s is 60. This means that, recall increases and missed objects are also substituted by better objects when s increases.

Let us now discuss the quality of the approximate classification when s is fixed to 100, and we vary the number k of objects retrieved for a class. Results of these experiments are shown in Figure 3. For each class considered (i.e. churches, temples, pyramids, paintings, and seascapes) we plot the recall and the EP vs the number of best matches k to a class. We note that both recall and EP are influenced by k: worse results are typically obtained when k increases. However, we can see that the reduction of the recall is in many cases minimal with respect to the increase of k, while the increase of the error on the position is more evident. This means that on average the approximate classification strategy is always able to find the same percentage of correct objects (almost stable recall), even if missed correct objects are replaced by worse objects (worsening EP). For instance, let us consider the class *pyramid*. Recall is around 0.9 for $k = 10$ and it goes to 0.75 when $k = 200$. That is, 1 out of 10 images is missed when we retrieve 10 objects, while a bit less than 3 out of 10 images are missed when we retrieve 200. The error in position is almost 0 when $k = 10$ and it is also negligible until $k = 100$. Thus, the ordering is practically maintained in the approximate result. When k increases more, the quality of ranking degrades. For instance, in case of $k = 200$, the error in position is about 0.00003. That is with a dataset of 1 million objects the average shift was of 30 positions, with respect to the exact rank.

We should also consider that the time required to perform exhaustive classification of the entire dataset, for a given class was 39 minutes, on average. Good approximate classification of the same dataset can be obtained on average in 1.5 minutes,thus the approximate classification is *more than one order of magnitude faster than exhaustive classification*.

B. User evaluation

Results discussed above were obtained comparing approximate classification against exhaustive classification algorithms, using the exhaustive classification as a ground truth. However, generally even the exhaustive classification presents some imprecisions, which can be evaluated when users are called to judge the result, or by using real ground truths. As we will see in the following, surprisingly, users do not see much difference between exact and approximate results. This means that the degradation from exact to approximate classification is purely mathematical, and it is not significantly perceived by users.

The test discussed in this section evaluates the difference of quality between the exhaustive top-k execution and the approximate top-k execution, as perceived by real users. To obtain this, we asked 5 students to blindly judge the results, obtained by the exhaustive and approximate classification, by selecting the good and the wrong images. Based on this, we computed the average precision using Equation 7. Results are shown in Figure 2 for various choices of the number of best matches k.

It can be seen that generally there is no significant difference between the precision of the exhaustive and the approximate classification, even thought the approximate classification is much faster. In fact, precision measured for the exhaustive and approximate classification has practically the same trend when k varies. In addition, generally the difference in precision, between the exhaustive and the approximate classification, is smaller than 10%.

A separate discussion is needed for the Seascape classifier. In the experiments discussed in previous section, results for the Seascape class were worse than all the other classes (see Figure 3). In fact, recall was always below 0.4. On the other hand, the user perceived precision of the approximate classification is very high and always above 0.8. It can be seen, also, that the exhaustive classification has also a precision above 0.9 in most cases. When the approximate classification is used, missed images are always substituted by other images that are deemed to be still good by human evaluators, offering an high precision. Therefore, approximation makes sense also in this case.

It is also worth mentioning that in one case of the tested classes, the approximate classification performed even better than the exhaustive one. In fact, it can be seen that for the Pyramids class, the curve of the approximate classification is always higher than that of the exhaustive one. This, we believe, is a further proof that no significant information is actually lost during the approximation: the lost information is mainly noisy information.

IX. Conclusions

Science is becoming data-dominated. New data-intensive computing paradigms are emerging that differ from the traditional techniques, where Big Data was not a fundamental issue [14]. We have presented an approximate technique for efficiently executing top-k classification tasks on very large datasets. The proposed technique is some orders of magnitude faster with respect to exhaustive classification and the accuracy of approximate results is very high.

The peculiarity of the proposed technique is that it is able to use one single index built in the input space to support top-k classification tasks on several classes defined using various kernels and kernel parameters. We discussed the properties that the kernel has to satisfy so that they can be used with the proposed technique and we have seen that many widely used kernels are in fact included.

Acknowledgements

This work was partially supported by the VISITO Tuscany POR CREO FESR 2007-2013 project, funded by the Regione Toscana.

References

[1] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge University Press, March 2000.

[2] A. Qamra and E. Y. Chang, “Using pivots to index for support vector machine queries,” in *CVDB ’05*, New York, NY, USA, 2005, pp. 59–64, ACM.

[3] N-Panda and E. Y. Chang, “Exploiting geometry for support vector machine indexing,” in *Proceedings of SIAM International Data Mining Conference, SDM*, 2005, pp. 322–333.

[4] S. Litayem, A. Joly, and N. Boujemaa, “Interactive objects retrieval with efficient boosting,” in *Proceedings of ACM Multimedia*, 2009, pp. 545–548.

[5] M. Crucianu, D. Estevez, V. Oria, and J.P. Tarel, “Speeding up active relevance feedback with approximate knn retrieval for hyperplane queries,” *Int. J. Imaging Syst. Technol.*, vol. 18, no. 2-3, pp. 150–159, 2008.

[6] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search - The Metric Space Approach*, vol. 32 of *Advances in Database Systems*, Springer, 2006.

[7] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[8] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ”nearest neighbor” meaningful?,” in *ICDT ’99, Proceedings*. 1999, vol. 1540 of *LNCS*, pp. 217–235, Springer.

[9] S. Boughorbel, J.P. Tarel, and N. Boujemaa, “Conditionally positive definite kernels for svm based image recognition,” in *IEEE ICME 2005*. July 2005, pp. 113–116, IEEE.

[10] C. Cortes, P. Haffner, and M. Mohri, “Rational kernels: Theory and algorithms,” *J. Mach. Learn. Res.*, vol. 5, pp. 1035–1062, 2004.

[11] P. Bolettieri, A. Esuli, F. Falchi, et al., “Enabling content-based image retrieval in very large digital libraries,” in *Second Workshop on VLDL*, 2009, pp. 43–50.

[12] G. Aamato and P. Savino, “Approximate similarity search in metric spaces using inverted files,” in *InfoScale ’08*. 2008, pp. 1–10, ICST.

[13] M. Batko, F. Falchi, C. Lucchese, et al., “Building a web-scale image similarity search system,” *Multimedia Tools and Applications*, vol. 47, no. 3, pp. 599–629, 2009.

[14] T. Hey, S. Tansley, and K. Tolle, Eds., *The Fourth Paradigm - Data Intensive Scientific Discovery*, Microsoft Res., 2009.