International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:1, 2013

On Speeding Up Support Vector Machines: Proximity Graphs *Versus* Random Sampling for Pre-Selection Condensation

Xiaohua Liu, Juan F. Beltran, Nishant Mohanchandra, and Godfried T. Toussaint

***Abstract*—**Support vector machines (SVMs) are considered to be the best machine learning algorithms for minimizing the predictive probability of misclassification. However, their drawback is that for large data sets the computation of the optimal decision boundary is a time consuming function of the size of the training set. Hence several methods have been proposed to speed up the SVM algorithm. Here three methods used to speed up the computation of the SVM classifiers are compared experimentally using a musical genre classification problem. The simplest method pre-selects a random sample of the data before the application of the SVM algorithm. Two additional methods use proximity graphs to pre-select data that are near the decision boundary. One uses *k*-Nearest Neighbor graphs and the other Relative Neighborhood Graphs to accomplish the task.

***Keywords*—**Machine learning, data mining, support vector machines, proximity graphs, relative-neighborhood graphs, *k*-nearest-neighbor graphs, random sampling, training data condensation.

I. Introduction

THIS paper is concerned with methods that attempt to reduce the computation time required to train support vector machines (SVMs) for applications in pattern classification tasks in the context of machine learning [38]. Support vector machines are considered to be among the best machine learning algorithms for minimizing the predictive probability of misclassification on a given problem [34]. However, they are often criticized for how time consuming the functions for computing the optimal decision boundary and making decisions on new patterns are, functions that depend on the size of the initial training set and the number of support vectors generated by the training algorithm, respectively. There have been several attempts at reducing these complexities. To reduce the number of support vectors obtained, the *data-post-processing* approach prunes the support vector set after the SVM classifier is built [12], [32]. However, since the bulk of the computational complexity results from building the classifier (training), these approaches are limited to reducing the testing time but not the training time. The *algorithmic* approach seeks to speed up the optimization procedure inherent in the SVM algorithm, either exactly or via approximation [13], [20], [22], [26]-[27], [31], [37]. One such method to reduce the complexity of building the SVM classifier, finds basis functions that maximize accuracy in an incremental way, permitting the user to stop this process once some limiting level of complexity has been reached [21]. A simpler, more straightforward, and direct approach to speed up SVMs is to use only a fraction of the training data in the optimization algorithm that builds the SVM classifier. There exists a plethora of these *data-pre-processing* (pre-selection) methods for choosing representative subsets of a data set. Comprehensive surveys of these techniques are also available [11], [35]. However, only few of these methods have been applied to SVM classification [7], [30], [42]. Lee and Mangasarian pre-select a small *random* sample of the data from which the SVM classifier is designed [24]. While selecting a random sample of the data, also called *bagging*, is very efficient computationally, intuition suggests that the randomness aspect of this procedure may not yield the best subset in terms of classification accuracy, for a given fractional size of the training data [5]. To address this concern, another class of methods selects a subset of the data that lie either near the boundary of the class-conditional distributions, or close to the decision boundaries between classes. The earliest algorithm for selecting points that lie near the decision boundary is Peter Hart's *condensed nearest neighbor* rule, which finds a *training-set-consistent* subset of the training data [17] A training-set-consistent subset of the data is one that classifies *all* the data correctly using the 1-nearest-neighbor decision rule [2]-[3]. Wang *et al*. select a subset of the data (which they call *key vectors*) that lie on the boundary of each class, using Parzen window estimates of the class-conditional probability density functions in the feature space [40]. Wang, Neskovic, & Cooper select a pattern near the decision boundary if its distance to a pattern from a different class is relatively small [39]. Nghi and Mai select data near the decision boundary by choosing for each data point, the nearest point from another class, and then iterating this process by exchanging the roles of the classes [29]. Koggalage and Halgamuge first use cluster analysis techniques to identify clusters in the data, and then identify *crisp clusters*, *i.e*., those clusters that have all their points belonging to the same class [23]. Finally they discard points that lie in the interiors of crisp clusters, on the grounds that they are unlikely to be support vectors. A similar approach by

X. Liu is with the Mathematics Department at New York University Abu Dhabi in Abu Dhabi, United Arab Emirates (e-mail: xl450@ nyu.edu).

J. F. Beltran is with the Computer Science Department at New York University Abu Dhabi in Abu Dhabi, United Arab Emirates (e-mail: jfb325@ nyu.edu).

N. Mohanchandra is with the Computer Science Department at New York University Abu Dhabi in Abu Dhabi, United Arab Emirates (e-mail: nm1345@ nyu.edu).

G. T. Toussaint is a Research Professor of Computer Science at New York University Abu Dhabi in Abu Dhabi, United Arab Emirates (e-mail: gt42@ nyu.edu).

Chen and Liu select points that are far from cluster centers [9]. Almeida *et al*. [1], and Li and Simske [25] also apply cluster analysis techniques to training set compression. An additional approach to pre-selecting training data makes use of proximity graphs, motivated by the fact that there appears to be a connection (not yet formally established) between support vectors and Gabriel graph neighbors [44]. Zhang and King (2002) used β-skeletons to reduce the size of the training set before computing the support vectors [43]. The β-skeletons are a general parameterized family of proximity graphs. For $\beta = 2$, the β-skeleton reduces to the relative neighborhood graph (RNG), and when $\beta = 1$, to the Gabriel graph (GG).

Here three methods used to speed up the computation of the SVM classifiers are compared experimentally using a music genre classification data set. Two methods use proximity graphs for pre-selecting points that are near the decision boundary. The first method uses the RNG, and the second method applies the *k*-nearest-neighbor (*k*-NN) graph, [10], [19], [36]. The RNG has been used successfully to reduce training set size in pattern classification tasks [4], [33]. Han et al. (2008) applied the RNG as a pre-selection method to speed up SVMs, but in comparing the resulting training times, testing times, and classification accuracies they neglected to take into account the time it takes to compute the RNG prior to pre-selection [16]. The *k*-NN graph has also been applied previously to speed up SVMs but with a complicated scoring function that determines which points are close to the decision boundary [30]. The *k*-NN method tested here on the other hand is very simple. These two methods are compared with pre-selection of a fractional random sample.

II. The Condensing Algorithms

A. Relative Neighborhood Condensation

The algorithm for condensing the training data with the Relative Neighborhood (RNG) algorithm deletes all the training samples (in parallel) that have the property that all their RNG neighbors belong to the same class as itself. It has been shown in earlier studies that this method of data condensation discards a large fraction of the data [33]. The RNG is defined as follows (refer to Fig. 1). Given two data points *a* and *b*, the *lune* of *a* and *b*, denoted by *Lune*(*a,b*), is the intersection of the two hyperspheres with centers at *a* and *b*, and radii equal to the distance between *a* and *b*. The two points *a* and *b* are *relative neighbors* provided that the *Lune*(*a,b*) contains no other point. In Fig. 1 *a* and *b* are not relative neighbors because point *c* is contained in *Lune*(*a,b*). The RNG is obtained by connecting with an edge all pairs of points that are relative neighbors of each other. A brute-force implementation of this condensing algorithm constructs the lune for each pair of points, and then tests every other point for inclusion in the lune, resulting in a running time complexity of $O(n^3)$, where *n* denotes the number of instances in the data set. A more clever implementation can be made to run in $O(n^{2+\varepsilon})$, where ε depends on the dimensionality of the data, but is usually relatively small [33]. The speedup algorithm is illustrated in Fig. 1. Rather than testing the emptiness of the lunes for all pairs of points, find all the relative neighbors of each point instead. In the first loop of the algorithm assume that all the relative neighbors of point *a* are to be computed. In the second loop of the program each point is tested in turn. Assume point *b* is tested. In the third loop all other points are tested for inclusion in *Lune*(*a,b*) as follows. First construct a hyperplane *H* through *b* and tangent to the hypersphere centered at *a*. When examining a point such as *d* for inclusion in *Lune*(*a,b*), first test whether it lies on the side of *H* that does not contain *a*. If this is the case then point *d* cannot possibly be contained in *Lune*(*a,b*), and it is not necessary to perform the lune-inclusion test in the third loop. However, the great savings comes, not from avoiding this test in the third loop, but rather from the fact that point *d* need not be tested as a candidate for a relative neighbor of *a* in the second loop of the program. This follows because the angle *abd* is greater than 90 degrees, implying that point *b* is contained in *Lune*(*a,d*), thus signaling that *d* cannot be a relative neighbor of *a*.

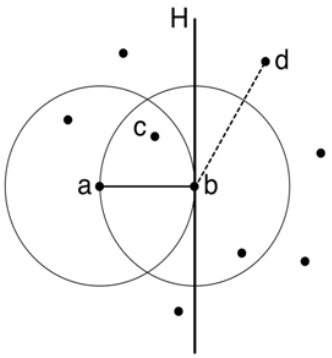

Fig. 1 Speeding up the computation of the RNG

This method is illustrated in Figs. 2-4. Fig. 2 shows a set of 50 data points in two dimensions, that belong to two classes, denoted by white and black squares.

Fig. 2 A training data set consisting of points from two classes

The relative neighborhood graph (RNG) of the data points in Fig. 2 is shown in Fig. 3, where those edges of the graph that connect points that belong to different classes are highlighted by dashed lines. The resulting condensed set of points is shown in Fig. 4.

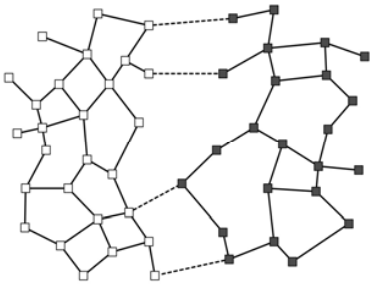

Fig. 3 The RNG of the data points of Fig. 2

Fig. 4 The RNG condensed set obtained from Fig. 3

B. k-Nearest Neighbor Condensation

The algorithm for condensing the training data via the *k*-Nearest Neighbor algorithm works as follows. First each data point X is marked if all its *k*-Nearest Neighbors belong to the same class as the class of X. Note that the marking is done in parallel for all the data. Then all the marked points are discarded. This approach is inspired by an algorithm due to Denis Wilson, for identifying points that lie near the decision boundary [41]. Wilson classified a point as far from the decision boundary if the *majority* of its *k* nearest neighbors classified it correctly. In this project 'majority' is replaced by 'all' in the voting scheme. This condensing algorithm runs in $O(kn^2)$ time using a straightforward implementation. Figs. 5-7 illustrate this process for 1-NN condensation. The 1-Nearest Neighbor graph of the points in Fig. 5 is shown in Fig. 6. In this graph, a point is connected by a directed edge (arrow) to its nearest neighbor.

Fig. 5 A training set consisting of two classes

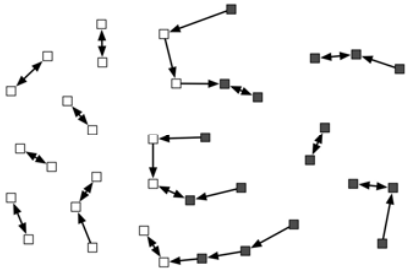

Fig. 6 The nearest neighbor graph of the points in Fig. 4

If an edge in the 1-Nearest Neighbor graph connects points that belong to the same class then these points are discarded from the data. The resulting condensed set obtained from Fig. 6, is shown in Fig. 7. Note that if the two classes are widely separated it may happen that no two points belonging to different classes are nearest neighbors of each other, resulting in the discarding of the entire data set. This happens for example with the set of points in Fig. 1. In such a case the value of *k* needs to be increased sufficiently to ensure that some data points from each class are kept.

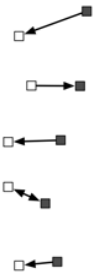

Fig. 7 The 1-NN condensed set obtained from Fig. 6

C. Random Sample Condensation

The third method for pre-selecting a small portion of the data is the simplest of all. It naively pre-selects a random sample of the data, and thus runs extremely efficiently in $O(n)$ time. Because of its simplicity this method is often used to compare how well more complex methods perform, and surprisingly there exists some evidence that it can perform quite well [39].

III. The Data

A. The Music Data

The music data consists of 23 pattern classes (music genres) all of which also fall into two larger classes (Western and Non-Western music). The Western genre consists of metal, pop, hiphop, funk, disco, rock, alternative, soul, electronica, religious, blues, country, reggae, jazz, and classical. The Non-Western genres are made up of China, Java, Arabic, India, Centralasia, Africa, and Japan. There are a total of 1641 instances, each with 40 real-valued tonal features related to the pitch class distribution, pitch range and musical scale employed, gamut and tuning system [15]. In the experiments

carried out here, 10% of the data was randomly set aside as the testing set, and the remaining 90% was used for training. All accuracy and timing outcomes were averaged over 10 trials in order to obtain standard deviations to serve as significance estimates of the resulting differences.

IV. Results

A. Phylogenetic Tree Clasification

In order to better understand the discrimination information contained in the features of the music data, and the resulting possible upper bounds on the classification accuracy of any decision rule, the dissimilarity of each genre was computed by first calculating the mean (also median) feature vector for each genre, and from that, the 1st-order Minkowski metric (L_1-norm) between each pair of mean (median) vectors. The resulting distance matrices were used to calculate the phylogenetic tress displayed in Figs. 8 and 9 using the *BioNJ* Algorithm [14], one of several algorithms embedded in the *SplitsTree-4* software package [6], [18]. Note that the distance between two genres in these trees is not the straight-line Euclidean distance between two genre nodes (leafs in the tree) but rather the shortest path (geodesic) between them along the branches in the tree. The tree is computed so as to match as closely as possible the corresponding distances in the distance matrix. The *SplitsTree-4* software package also employs an embedded graph-drawing algorithm that draws the tree nicely. However one can manually reposition the tree, much like a linkage that allows rotation of its joints, to suit one's purpose without changing any of the distances involved. The tree in Fig. 8, calculated with the mean feature vector for each genre, shows a marked clustering of the genres into two clusters. The cluster at the top contains all the Non-Western genres but includes five Western genres interspersed among them: blues, country, folk, jazz, and classical, with classical lying furthest from the group. The cluster at the bottom consists of all Western genres, with metal as the distant outlier of the group.

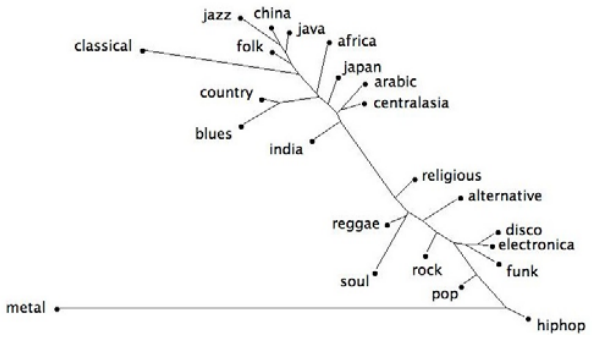

Fig. 8 *BioNJ* tree for the music data using mean vectors

Since the median is a more robust estimate of centrality than the mean the *BioNJ* tree was also computed with the median vectors (Fig. 9). Globally the clustering of this tree exhibits general agreement with the clustering obtained with the mean-vector tree of Fig. 8. However, there are some notable differences: here reggae and India moved closer to the Non-Western cluster at the top, whereas country and blues moved closer to the Western cluster at the bottom.

One way to classify the genres using these phylogenetic trees is to create a sub-tree by cutting off a branch of the tree (deleting one of the edges). In cladistic analysis the resulting sub-tree is called a *clade* [28]. In Fig. 8 the best clade (obtained by cutting between India and reggae) contains five misclassified groups: blues, country, classical, folk and jazz, which are misclassified as Non-Western. On the other hand, the best clade in Fig. 9 (obtained by cutting between folk and Africa) yields only three misclassified rhythms: classical and jazz, wrongly grouped with non-Western, and Japan, wrongly grouped with Western. Hence, the median feature vector tree appears to capture more discrimination information for these two general genres than the mean feature vector tree.

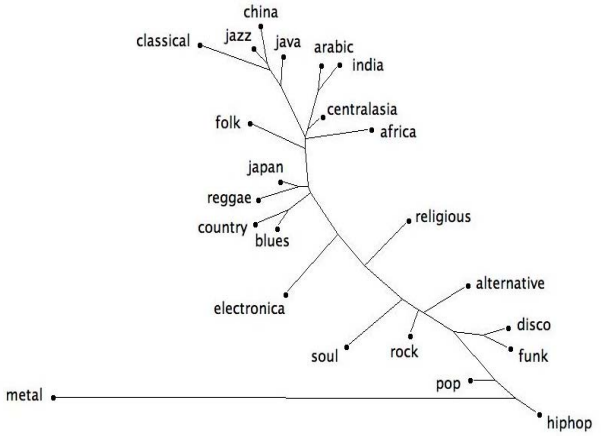

Fig. 9 *BioNJ* tree for the music data using median vectors

B. SVM Speedup Methods

The standard SVM training algorithm runs in $O(n^3)$ time and $O(n^2)$ space, where n is the size of the training set [37]. The sequential minimal optimization (SMO) algorithm for SVM training, as implemented in the Weka package used here, appears to run in time-complexity somewhere between $O(n)$ and $O(n^{2.2})$, depending on the structure of the training data [31]. To determine empirically how fast the computation time of SMO grows as a function of the size of the data, experiments were performed with random subsets of the training data of varying size. The results are shown in Figs. 10 and 11. Fig. 10 suggests that the time complexity for this music data does grow at least quadratically as the size of the training data increases. Furthermore, as Fig. 11 shows, good accuracy (above 87%) is achieved only when at least 75% of the data are used.

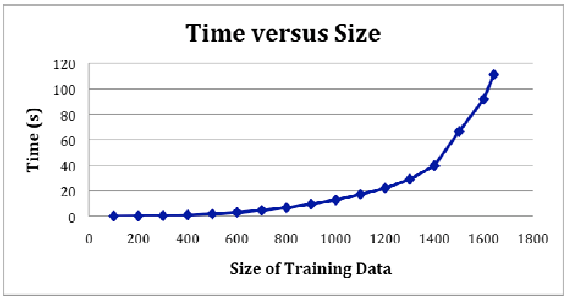

Fig. 10 Time versus data size for the Weka SMO

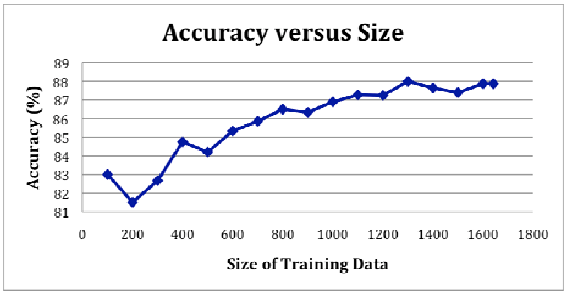

Fig. 11 Accuracy versus data size for the Weka SMO

The *k*-NN rule achieves a top mean accuracy of 87.6% for *k* = 3. The SMO support-vector machine classifier achieved an accuracy of 86.5% in the experiments performed by Gomez and Herrera [15], but they reported no standard deviations and no timing results. Replicating their experiment using the same parameters [8] (complexity = 1.5, polynomial kernel exponent = 1.5), Toussaint and Berzan [34] obtained a significantly higher mean accuracy of 89.4%. In the experiments reported here the same parameters: complexity (1.5) and polynomial kernel exponent (1.5), with Weka SMO, were used in an environment consisting of an Intel Core i5 CPU M450 @ 2.40GHz, with a RAM of 2.00 GB and a Windows 7-64 bit operating system.

TABLE I
TIMES AND ACCURACIES FOR THE EDITING METHODS TESTED

Edit Method	Edit Time	Data Size	Training Time	Testing Time	Accuracy
No Edit	0	1641	5.7 ±0.45	0.15 ±0.03	89.4 ±1.9
1-NN	73.9 ±0.52	412	0.57 ±0.15	0.08 ±0.01	80.6 ±3.0
3-NN	75.5 ±2.11	794	2.0 ±0.55	0.14 ±0.04	87.6 ±2.1
RNG	895 ±267	813	1.95 ±0.47	0.13 ±0.03	87.5 ±1.7
Random (25%)	< 0.01	~400	0.39±0.08	0.06 ±0.01	86.5 ±1.9
Random (50%)	< 0.01	~800	1.4 ±0.19	0.095 ±0.01	88.2 ±1.8

When comparing the effects of the different editing schemes on the speed-up of the resulting SVM, there are four different times that are of interest: (1) the training time taken by the SVM algorithm with the edited set, (2) the testing time taken by the SVM algorithm with the edited set, (3) the time taken to edit (condense) the original training data, and (4) the total time, *i.e.*, the sum of (1), (2), and (3). The results, along with ± one standard deviation calculated from the ten runs, are summarized in Table I for the six methods listed in column 1. No Edit refers to the application of SVM training on the entire data consisting of 1641 instances. The *k*-NN editing scheme was implemented with two values of *k*, *k* = 1 (denoted by 1-NN) and *k* = 3 (denoted by 3-NN). Values of *k* greater than 3 were not tried because the best previous classification accuracies obtained with SMO were for *k* = 3 [15]. The editing with the relative neighborhood graph is labeled RNG. For the random sampling approach two values for the size of the reduced set were selected: 25% or about 400 instances, and 50%, or about 800 instances. These values were selected for comparison with the proximity graph methods because the 1-NN editing scheme kept about 25% of the data (412), and both the 3-NN and RNG editing schemes kept about 50% of the data (794 and 813, respectively).

For easy visualization and comparison, the results are also displayed as graphs in Figs. 12-17. From Fig. 12 it is clear that all the methods yield a considerable decrease in the training time taken by the Weka SMO algorithm when compared to no editing at all. Furthermore, the training times of SMO with the random sample methods are significantly lower than with the 3-NN and RNG edited sets. The fastest training time is obtained with a random sample of size 25% of the entire data.

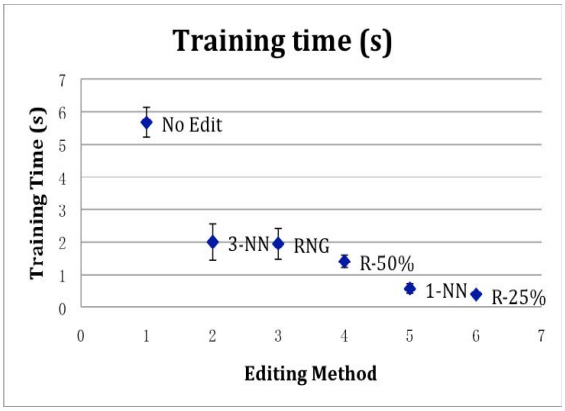

Fig. 12 Training time obtained for the different editing methods

The testing time results, illustrated in the graph of Fig. 13 tell a slightly different story. The RNG and 3-NN editing schemes yield similar average times with relatively large standard deviations, thus providing no advantage over no editing at all. However, 1-NN condensation and random sampling have relatively small standard deviations, and are thus statistically significantly faster than the other three methods. The fastest of all the methods is still random sampling with 25% of the data, which is statistically significantly faster than all the other methods.

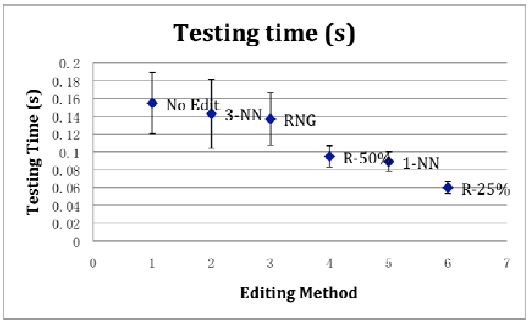

Fig. 13 Testing time obtained for the different editing methods

The results on training and testing times of the SMO experiments with the pre-selected condensed training sets agree with previous results obtained by other researchers with other methods of pre-selection. However, previous studies neglected to take into account the time taken by the pre-selection algorithms [16]. Although it is an interesting result that the training time of SVM algorithms can be reduced to acceptable levels with smaller training sets, without significantly degrading the accuracy, the practical reality necessitates taking the pre-selection time into account. This time must be added to the SVM training time to determine if such an approach is practical. Hence the most meaningful time to use for comparing the various methods is the total time = pre-selection time + SVM training time on the reduced set + testing time with the SVM. This total time is illustrated in the graph in Fig. 14. It is patently clear that the pre-selection time for the RNG method (895 seconds) stands out above all others so dramatically that it renders the method useless compared to the others. Furthermore, the 1-NN and 3-NN condensed sets, although considerably faster than the RNG condensed set, are still much slower than no editing at all or random sampling.

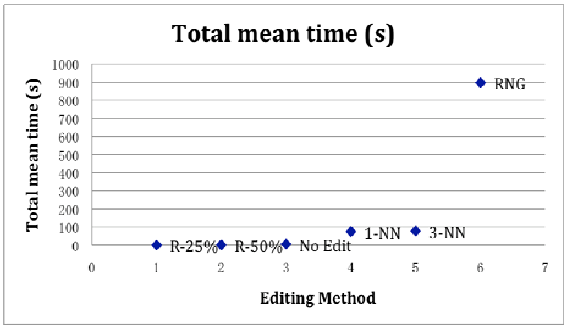

Fig. 14 Total time obtained for the different editing methods

From Fig. 14 it is difficult to compare the latter three methods because their values all appear to be zero. Therefore these three times are displayed in isolation in Fig. 15, where it becomes clear that the total time for random sampling is much smaller than the total time for no editing at all.

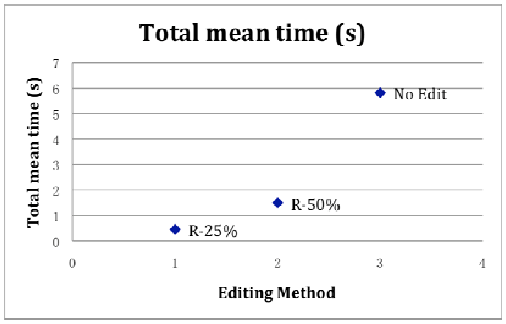

Fig. 15 Total time obtained for random sampling and no editing

One can justify the use of pre-selection methods that are computationally efficient only if the resulting accuracy is not significantly downgraded. Clearly, one may discard almost all the data and obtain very fast computation times if one is willing to tolerate low accuracies. Therefore the previous discussion must be examined in the context of the resulting accuracies. These results are visually displayed in Fig. 16. Only one result is manifestly clear here: the accuracy of the SVM with the 1-NN edited set is significantly worse than with all the other editing schemes. The mean accuracies with the other editing schemes range between 89.4% without any editing, to 86.5% with 25% random sampling. Although there is a general trend evident from the mean values: No Edit > R-50% > 3-NN > RNG > R-25%, these differences are not statistically significantly different.

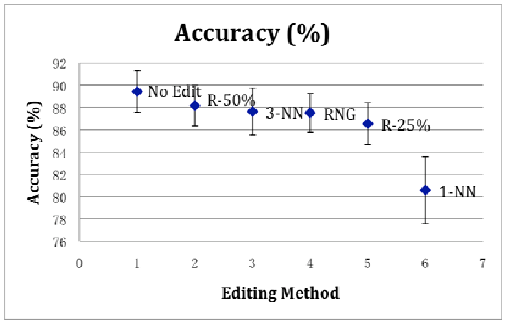

Fig. 16 Accuracy obtained with the different editing methods

In order to get an idea of how much data may be discarded by random sampling in order to match the accuracy of the 1-NN editing method, ten runs were carried out by selecting 100 random points from the 1641 data points. This corresponds to keeping only 6.1% of the data. The results for all ten runs are shown in Fig. 17. The average of these results is 81% ±3.06%, which matches the accuracy results of 1-NN editing. However, 1-NN editing keeps 412 data points, again demonstrating the power of random sampling over the "intelligent" 1-NN algorithm that tries to keep points near the decision boundary.

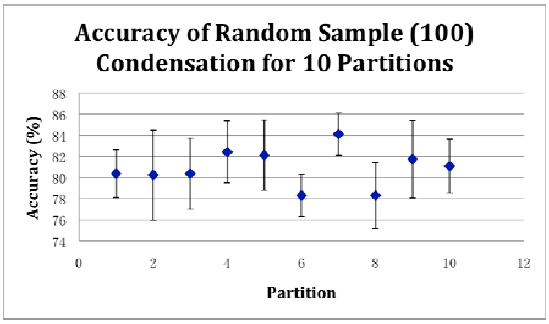

Fig. 17 Accuracy of the ten partitions with 100-point random pre-selection samples

V. Conclusion

In spite of all the previous successes of the application of proximity graphs in a variety of computer science problems in areas ranging from decision theory to shape analysis and computer vision to machine learning [19], [35], the results obtained here indicate that proximity graphs appear to be useless for speeding up support vector machines when they are used to preselect subsets of the training data, and when the computation time for this task is also taken into account. Indeed, the preliminary results obtained in this study suggest that, contrary to intuition, and in agreement with the results of [39], naive random sampling appears to be the method of choice for this application: it is the simplest method, computationally extremely efficient, and retains good accuracy compared with using the entire training data.

The size of the music data utilized in the experiments carried out in this pilot project (1641) is not extremely large by today's standards. Hence it would be interesting to compare random sampling with no editing, for much larger data sets (say at least an order of magnitude larger) to determine how well random sampling scales in terms of both computational speed and classification accuracy.

Acknowledgment

This research was supported by a grant from the Provost's Office of New York University Abu Dhabi, through the Faculty of Science, in Abu Dhabi, The United Arab Emirates. The authors thank Professor Emilia Gomez of the Music Technology Group at Universitat Pompeu Fabra in Barcelona, Spain, for providing them the music data, and to Constantin Berzan of Tufts University in Medford, MA, USA, for making his RNG program available.

References

[1] M.B. Almeida, A.P. Braga, and J.P. Braga, "SVM-KM: speeding SVMs learning with a priori cluster selection and k-means," In: *Proceedings of the 6th Brazilian Symposium on Neural Networks,* pp. 162–167, 2000.

[2] F. Angiulli, "Fast nearest neighbor condensation for large data sets classification," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 11, pp. 1450–1464, Nov. 2007.

[3] F. Angiulli and A. Astorino, "Scaling up support vector machines using nearest neighbor condensation," *IEEE Transactions on Neural Networks*, vol. 21, no. 2, pp. 351-357, February 2010.

[4] B. Bhattacharya, K. Mukherjee and G.T. Toussaint, "Geometric decision rules for instance-based learning algorithms," *Proc. Pattern Recognition and Machine Intelligence: First International Conference*, S. K. Pal et al., (Eds.): LNCS 3776, Kolkata, India, pp. 60-69, December 20-22, 2005.

[5] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, 1996.

[6] D. Bryant and V. Moulton, "*NeighborNet*: An agglomerative algorithm for the construction of phylogenetic networks," *Molecular Biology and Evolution*, vol. 21, no. 2, pp. 255–265, 2004.

[7] C.J.C. Burges and B. Scholkopf, "Improving the accuracy and speed of support vector learning machines," In *Proceedings of the 9th NIPS Conference*, pages 375–381, 1997.

[8] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing multiple parameters for support vector machines," *Machine Learning*, vol. 46, pp. 131–159, 2002.

[9] J. Chen and C.-L. Liu, "Fast multi-class sample reduction for speeding up support vector machines," *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing*, Beijing, China, September 18-21, 2011.

[10] T.M. Cover and P.E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27, 1967.

[11] B.V. Dasarathi, "Tandem fusion of nearest neighbor editing and condensing algorithms - data dimensionality effects," *Proceedings of the 15th International Conference on Pattern Recognition*, vol. 2, pp. 692-695, 2000.

[12] T. Downs, K.E. Gates, and A. Masters, "Exact simplification of support vector solutions," *Journal of Machine Learning Research*, vol. 2, pp. 293–297, 2001.

[13] S. Fine and K. Scheinberg, "E cient SVM training using low-rank kernel representation, *Journal of Machine Learning Research,* pp. 243–264, 2009.

[14] O. Gascuel, "*BIONJ*: an improved version of the NJ algorithm based on a simple model of sequence data," *Molecular Biology and Evolution,* vol. 14, no. 7, pp. 685-695, 1997.

[15] E. Gomez and P. Herrera, "Comparative analysis of music recordings from Western and Non-Western traditions by automatic tonal feature extraction," *Empirical Musicology Review*, vol. 3, pp. 140-156, 2008.

[16] D. Han, C. Han, Y. Yang, Y. Liu, and W. Mao, "Pre-extracting method for SVM classification based on the non-parametric *K*-NN rule," *Proceedings of the 19th International Conference on Pattern Recognition*, pp. 1-4, Dec. 8-11, 2008.

[17] P.E. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, pp. 515–516, 1968.

[18] D.H. Huson, "*SplitsTree*: A program for analyzing and visualizing evolutionary data," *Bioinformatics*, vol. 14, no. 10, pp. 68–73, 1998.

[19] J.W. Jaromczyk and G.T. Toussaint, "Relative neighborhood graphs and their relatives," *Proc. of the IEEE*, vol. 80, no. 9, September, pp. 1502-1517, 1992.

[20] T. Joachims, "Making large-scale SVM learning practical," Advances, In: *Kernel Methods - Support Vector Learning*, MIT-Press, Cambridge, MA, 1999.

[21] S.S. Keerthi, O. Chapelle, and D. DeCoste, "Building support vector machines with reduced classifier complexity," *Journal of Machine Learning Research*, vol. 7, pp. 1493–1515, 2006.

[22] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to Platt's SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, pp. 637–649, 2001.

[23] R. Koggalage and S. Halgamuge, "Reducing the number of training samples for fast support vector machine classification," *Neural Information Processing – Letters and Reviews*, vol. 2, no. 3, pp. 57-65, March 2004.

[24] Y.J. Lee and O.L. Mangasarian, "RSVM: Reduced support vector machines," In *Proceedings of the First SIAM International Conference on Data Mining*, SIAM, Chicago, April 5-7, 2001.

[25] D. Li and S. Simske, "Training set compression by incremental clustering," *Journal of Pattern Recognition Research*, vol. 1, pp. 56-64, 2011.

[26] X. Liang, R.-C. Chen, and X. Guo, "Pruning support vector machines without altering performances," *IEEE Transactions on Neural Networks*, vol. 19, no. 10, pp. 1792-1803, October 2008.

[27] M.E. Mavroforakis and S. Theodoridis, "A geometric approach to support vector machine (SVM) classification," *IEEE Transactions on Neural Networks,* vol. 17, no. 3, pp. 671-682, May 2006.

[28] E. Mayr, "Cladistic analysis or cladistic classification?" *Zeitschrift fuer Zoologische Systematik und Evolutionsforschung,* vol. 12, pp. 94-128, 1974.
[29] D.H. Nghi and L.C. Mai, "Training data selection for support vector machines model," *Proceedings of the International Conference on Information and Electronics Engineering,* vol. 6, Press, Singapore, 2011.
[30] N. Panda, E. Y. Chang, and G. Wu, "Concept boundary detection for speeding up SVMs," *Proceedings of the 23 International Conference on Machine Learning,* Pittsburgh, PA, 2006.
[31] J.C. Platt, "Fast training of support vector machines using sequential minimal optimization," In B. Scholkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning,* Cambridge, MA, MIT Press, 1998.
[32] T. Thies and F. Weber, "Optimal reduced-set vectors for support vector machines with a quadratic kernel," *Neural Computation*, vol. 16, pp. 1769–1777, 2004.
[33] G.T. Toussaint, B. K. Bhattacharya, and R. S. Poulsen, "The application of Voronoi diagrams to nonparametric decision rules," *Proc. Computer Science and Statistics: 16th Symposium on the Interface*, Atlanta, Georgia, March 14-16,1984, Published by North-Holland in 1985, Amsterdam, L. Billard, Ed., pp. 97-108.
[34] G.T. Toussaint and C. Berzan, "Proximity-graph instance-based learning, support vector machines, and high dimensionality: An empirical comparison." *Proceedings of the Eighth International Conference on Machine Learning and Data Mining*, July 16-19, 2012, Berlin, Germany. P. Perner (Ed.): MLDM 2012, LNAI 7376, pp. 222–236, 2012. Springer-Verlag Berlin Heidelberg.
[35] G.T. Toussaint, "Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining," *International Journal of Computational Geometry and Applications*, vol. 15, April 2005, pp. 101-150.
[36] G.T. Toussaint, "The relative neighborhood graph of a finite planar set," *Pattern Recognition*, vol. 12, pp. 261-268, 1980.
[37] I. W. Tsang, J. T. Kwok, and P.-M. Cheung, "Core vector machines: Fast SVM training on very large data sets," *Journal of Machine Learning Research*, vol. 6, pp. 363–392, 2005.
[38] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, NY, 1995.
[39] J. Wang, P. Neskovic, L. N. Cooper, "Selecting data for fast support vector machine training," *Studies in Computational Intelligence (SCI),* vol. 35, pp. 61–84, 2007.
[40] Y. Wang, C.G. Zhou, Y.X. Huang, Y.C. Liang, and X.W. Yang, "A boundary method to speed up training support vector machines," In: G. R. Liu *et al*. (eds), *Computational Methods*, Springer, Printed in the Netherlands, pp. 1209–1213, 2006.
[41] D.L. Wilson, "Asymptotic properties of nearest neighbor rules using edited-data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, pp. 408–421, 1973.
[42] M.H. Yang and N. Ahuja, "A geometric approach to train support vector machines," In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* 2000 (CVPR 2000), June, Hilton Head Island, pp. 430–437, 2000.
[43] W. Zhang and I. King, "Locating support vectors via β-skeleton technique," In: *Proceedings of the International Conference on Neural Information Processing* (ICONIP*),* pp. 1423–1427, 2002.
[44] W. Zhang and I. King, "A study of the relationship between support vector machine and Gabriel graph." In: *Proceedings of the IEEE International Joint Conference on Neural Networks*, IJCNN, Honolulu, vol. 1, pp. 239–244, 2002.