Evolution of Kernels for Support Vector Machine Classification on Large Datasets

Luis Carlos Padierna, Martín Carpio, Rosario Baltazar, Héctor José Puga and Héctor Joaquín Fraire

Abstract Kernel selection is a main factor in the designing of support vector machines. Evolutionary techniques have been applied to select the fittest kernel for specific classification problems. However, technical issues emerge when attempting to apply this methodology to deal with large datasets. On the other hand, a new method for improving the training time of support vector machines was recently developed. In this chapter, the new method is integrated in a kernel evolution scheme. Ten benchmark datasets are tested. Results indicate that the new method speeds up the evolution process when datasets are greater than 1000 instances.

1 Introduction

Support Vector Machines (SVMs) have been widely used for pattern classification. Key design aspects for achieving high performance rates in SVM classification involve: selection of the quadratic programming solver, kernel parameter tuning and kernel selection [1–3].

L.C. Padierna · M. Carpio · R. Baltazar · H.J. Puga (✉)
Tecnológico Nacional de México-Instituto Tecnológico de León, Leon, Mexico
e-mail: pugahector@yahoo.com

L.C. Padierna
e-mail: luiscarlos.padierna@itleon.edu.mx

M. Carpio
e-mail: juanmartin.carpio@itleon.edu.mx

R. Baltazar
e-mail: r.baltazar@ieee.org

H.J. Fraire
Tecnológico Nacional de México-Instituto Tecnológico de Cd. Madero, Cd. Madero, Mexico
e-mail: automatas2002@yahoo.com.mx

P. Melin et al. (eds.), *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization*,
Studies in Computational Intelligence 601, DOI 10.1007/978-3-319-17747-2_10

Kernel selection is useful when training an SVM with non-linearly separable data. The reason is that a kernel is generally a non-linear function that maps the original input space into a high-dimensional dot-product feature space in order to enhance linear separability [4]. Until now, there not exists a systematic way to choose the fittest kernel for a given dataset [5].

Furthermore, one single kernel could be not enough to reach good generalization rates. A current trend consists in combining two or more kernels with the intention to increase the generalization capability of an SVM [6]. This strategy is called Multiple Kernel SVM [2].

Recent approaches had considered evolutionary strategies to automatically choose a multiple kernel that best fit to a specific dataset [1, 7]. Approximate methods such as evolutionary strategies are needed since the problem of finding the best multiple kernel for a specific dataset is NP-complete [8]. However, these strategies involve prohibitive computational costs as the size of the dataset increases.

It is known that training a standard SVM has a complexity between $O(n^2)$ and $O(n^3)$ where n is the number of input vectors [3, 9]. Furthermore, the size of the associated Gram matrix is $n \times n$ (see Sect. 2.2), therefore, the cost in time and space that should be paid in every evaluation of the fitness function when evolving kernels is high.

In this chapter a new method is integrated into an evolutionary scheme in order to prove the hypothesis that, by applying it to a dataset, is possible to reduce the computational burden inherent to the evolution process and, in consequence, to accelerate the evolution of kernels for large datasets.

Section 2 provides fundamental definitions and a brief description of the techniques applied in this work. The evolutionary scheme integrating the new method, the materials and configurations used to carry out experiments are established in Sect. 3. Finally, a discussion about main results, future research guidelines and conclusions are provided in Sect. 4.

2 Theoretical Basis

In this section three methods are briefly described, namely, C-SVM, Genetic Programming (GP) and Decision Tree Support Vector Machines (DTSVM). In addition, some fundamental concepts are given in logical order.

2.1 Support Vector Machines

SVMs are non-probabilistic binary classifiers that can be used to construct a hyperplane to separate data into one of two classes. Its formulation is as follows [4, 10]:

Given a training dataset $D = \{x_{\mathrm{i}}, y_{\mathrm{i}}\}_{i=1}^{m}$ where $x_{\mathrm{i}} \in X, X \subset R^n, y_{\mathrm{i}} \in \{+1, -1\}$, SVM classifies with an optimal separating hyperplane, which is given by:

$$h(x) = w^{\mathrm{T}}x + b \tag{1}$$

When working with data non-linearly separable, this hyperplane is obtained by solving the following quadratic programming problem:

$$\begin{aligned} &\min\left(\frac{1}{2}w^{\mathrm{T}}w + C\sum_{\mathrm{i=1}}^{\mathrm{m}}\xi_{\mathrm{i}}\right) \\ &\text{s.t. } y_{\mathrm{i}}(w^{\mathrm{T}}x_{\mathrm{i}} + b) \geq 1 - \xi_{\mathrm{i}}, \xi_{\mathrm{i}} > 0 \quad \text{for } i = 1, \ldots, \mathrm{m} \end{aligned} \tag{2}$$

where ξ_i are slack variables to tolerate miss classifications and $C > 0$ is a regularization parameter. Introducing the nonnegative Lagrange multipliers α and β and following the Karush-Kuhn-Tucker conditions:

$$\nabla_{\mathrm{w}}L = w - \sum_{\mathrm{i=1}}^{\mathrm{m}}\alpha_{\mathrm{i}}y_{\mathrm{i}}x_{\mathrm{i}} = 0 \tag{3}$$

$$\nabla_{\mathrm{b}}L = -\sum_{\mathrm{i=1}}^{\mathrm{m}}\alpha_{\mathrm{i}}y_{\mathrm{i}} = 0 \tag{4}$$

$$\nabla_{\xi_{\mathrm{n}}}L = C - \alpha_{\mathrm{i}} - \beta_i = 0 \tag{5}$$

the problem (2) can be proved to be equivalent to the following dual problem

$$\begin{aligned} &\text{Max } L(\alpha) = \sum_{\mathrm{i=1}}^{\mathrm{M}}\alpha_{\mathrm{i}} - \frac{1}{2}\sum_{\mathrm{i=1}}^{\mathrm{m}}\sum_{\mathrm{j=1}}^{\mathrm{m}}y_{\mathrm{i}}y_{\mathrm{j}}\alpha_{\mathrm{i}}\alpha_{\mathrm{j}}k(x_{\mathrm{i}}, x_{\mathrm{j}}) \\ &\text{s.t. } C \geq \alpha_{\mathrm{i}} \geq 0 \quad \forall \mathrm{i} = 1, \ldots, \mathrm{m} \\ &\text{and } \sum_{\mathrm{i=1}}^{\mathrm{m}}\alpha_{\mathrm{i}}y_{\mathrm{i}} = 0 \end{aligned} \tag{6}$$

This expression is considered the standard version of an SVM and is called C-SVM.

2.2 Kernel, Multiple Kernel and Gram Matrix

Kernel A function $\mathrm{K}(x, x')$ defined on $R^n \times R^n$ is called a kernel on $R^n \times R^n$ or kernel briefly if there exists a map ϕ from the space R^n to the Hilbert space

Table 1 Some common kernel functions

Kernel	$K(x,x') =$	Kernel	$K(x,x') =$
Lineal	$x^{\mathrm{T}}x'$	Powered	$-\Vert x-x'\Vert^{\beta}\ 0<\beta\leq 1$
Polynomial	$(\sigma \times x^{T}x' + r)^{d}$	Log	$-\log\left(1+x-x'\Vert^{\beta}\right)\ 0<\beta\leq 1$
RBF	$e^{\left(-\frac{\Vert x-x'\Vert^2}{\sigma^2}\right)}$	Generalized Gaussian	$e^{-(x-x')^{\mathrm{T}}\mathrm{A}(x-x')}$ where A is a symmetric PD matrix
Sigmoid	$\tanh(\sigma \times x^{\mathrm{T}}x' + r)$	Hybrid	$e^{-\frac{\Vert x-x'\Vert^2}{\sigma^2}} \times (\tau + x^{\mathrm{T}}x')^{d}$

$\phi : R^n \rightarrow \mathcal{H}$ such that $\mathrm{K}(x,x') = (\phi(x)\cdot\phi(x'))$ where $(\cdot)$ denotes the inner product of space $\mathcal{H}$ [11]. Some common kernel functions are shown in Table 1 [12].

Multiple Kernel Is denoted as $\mathrm{K}_{\eta}(x_{\mathrm{i}},x_{\mathrm{j}}) = f_{\eta}\left(\left\{\mathrm{K}_{\mathrm{m}}\left(x_{\mathrm{i}}^{\mathrm{m}},x_{\mathrm{j}}^{\mathrm{m}}\right)\right\}_{m=1}^{P}\right)$ where the combination function $f_{\eta}\colon R^{P} \rightarrow R$, can be a linear or a nonlinear function and η parameterizes the combination function. The more common implementation is:

$$\mathrm{K}_{\eta}(x_{\mathrm{i}},x_{\mathrm{j}}) = f_{\eta}\left(\left\{\mathrm{K}_{\mathrm{m}}\left(x_{\mathrm{i}}^{\mathrm{m}},x_{\mathrm{j}}^{\mathrm{m}}\right)\right\}_{m=1}^{P}|\eta\right) \tag{7}$$

where the parameters are used to combine a set of predefined kernels (i.e., the kernel functions and corresponding kernel parameters are known before training) [2]. To implement a multiple kernel in an SVM, the requirement is that it fulfills the Mercer conditions [13]:

$$\begin{gathered} K(x,x') = \sum_{\mathrm{i}}^{\infty} a_{\mathrm{i}}\varphi_{\mathrm{i}}(x)\varphi_{\mathrm{i}}(x'), \quad a_{\mathrm{i}} > 0 \\ \int_a^b\int_a^b K(x,x')g(x)g(x')\mathrm{d}x\mathrm{d}x' > 0 \end{gathered} \tag{8}$$

where the variables x and x' are defined in the closed square $a \leq x \leq b$, $a \leq x' \leq b$ and $g(x)$ and $g(x')$ are any function continuous in (a,b).

Gram Matrix For a function $\mathrm{K}(x,x')\colon R^n \times R^n \rightarrow R$ and l points $x_1,\ldots,x_l \in R^n$, the $l \times l$ matrix G, of which the i-th row and the j-th column element is $\mathrm{G}_{\mathrm{ij}} = \mathrm{K}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)$, is called the Gram matrix of the function $\mathrm{K}(x,x')$ with respect to $x_1,\ldots,x_l$ [11].

2.3 *Genetic Programming*

This paradigm can search the space of possible computer programs for an individual computer program that is highly fit in solving the problem at hand. Genetic

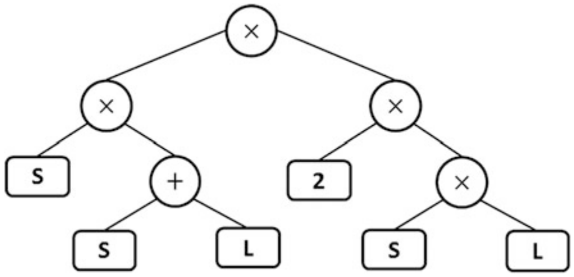


Fig. 1 Multiple kernel represented as a tree

Programming (GP) has been shown to be capable of inducing programs for fields as optimal control, planning, symbolic regression, automatic programming, and pattern classification among others. The induction is a result of the combination of an efficient learning procedure and expressive symbolic representations [14].

The program structure is closely related to a fitness function that guides certain evolution process. In the case of kernel evolution, tree data structures are commonly used for encoding programs as chromosomes. GP has been applied to find optimized kernel functions for SVM classification [1, 7]. In Fig. 1 a multiple kernel encoded as a tree is exemplified and correspond to the expression:

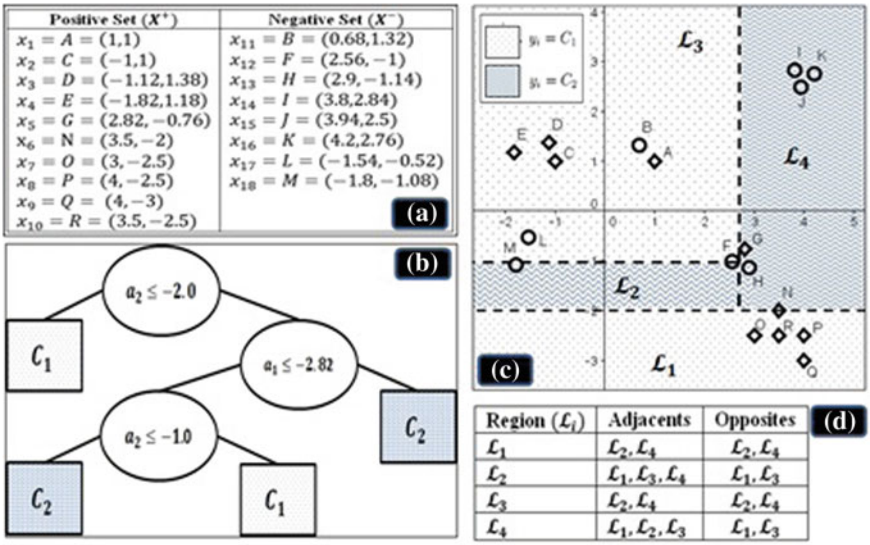
$$\mathrm{K}_{\eta}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)=\left(\mathrm{K_S}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)+\mathrm{K_L}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)\right)\left(2\times\mathrm{K_L}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)\times\mathrm{K_S^2}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)\right) \qquad (9)$$

where $S=\mathrm{K_S}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)$ and $L=\mathrm{K_L}\left(x_{\mathrm{i}},x_{\mathrm{j}}\right)$ are the Sigmoid and Linear Kernel, respectively.

For kernel construction the next three steps are suggested: find out basic kernels, find out the operations keeping kernels and construct kernels from basic kernels applying operations [11]. In this work, the basic kernels are four: Linear, RBF, Sigmoid and Polynomial. Two keeping kernels operations are considered, addition $(+)$ and multiplication $(\times)$. GP is used to construct kernels from basic kernels applying operations.

2.4 Decision Tree Support Vector Machine

Decision Tree Support Vector Machine (DTSVM) is a method for data reduction and was recently proposed in [3]. DTSVM aims to build a subset (X_R) of original set (X) such that X_R be much smaller than X, i.e., $X_R \subset X$: $|X_R| \ll |X|$. To obtain such a subset, the C4.5 algorithm is used to derive a Decision Tree (DT) which partitions the input space into regions with low entropy. Then, adjacent regions with opposite class label are detected. Finally, a Fisher Linear Discriminant (FLD) is applied to choose a proportion of elements nearest to the decision boundary whose could be support vectors. Figure 2 shows an illustration of the method.



Positive Set (X^+)	Negative Set (X^-)
$x_1 = A = (1,1)$	$x_{11} = B = (0.68,1.32)$
$x_2 = C = (-1,1)$	$x_{12} = F = (2.56,-1)$
$x_3 = D = (-1.12,1.38)$	$x_{13} = H = (2.9,-1.14)$
$x_4 = E = (-1.82,1.18)$	$x_{14} = I = (3.8,2.84)$
$x_5 = G = (2.82,-0.76)$	$x_{15} = J = (3.94,2.5)$
$x_6 = N = (3.5,-2)$	$x_{16} = K = (4.2,2.76)$
$x_7 = O = (3,-2.5)$	$x_{17} = L = (-1.54,-0.52)$
$x_8 = P = (4,-2.5)$	$x_{18} = M = (-1.8,-1.08)$
$x_9 = Q = (4,-3)$	
$x_{10} = R = (3.5,-2.5)$	

Region ($\mathcal{L}_i$)	Adjacents	Opposites
$\mathcal{L}_1$	$\mathcal{L}_2, \mathcal{L}_4$	$\mathcal{L}_2, \mathcal{L}_4$
$\mathcal{L}_2$	$\mathcal{L}_1, \mathcal{L}_3, \mathcal{L}_4$	$\mathcal{L}_1, \mathcal{L}_3$
$\mathcal{L}_3$	$\mathcal{L}_2, \mathcal{L}_4$	$\mathcal{L}_2, \mathcal{L}_4$
$\mathcal{L}_4$	$\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$	$\mathcal{L}_1, \mathcal{L}_3$

Fig. 2 Numerical example of DTSVM method. **a** Original dataset. **b** Induced decision tree from dataset. **c** Partition derived from DT. **d** Opposite regions to consider in FLD

In Fig. 2, a two class problem in R^2 space is presented. The original dataset (X) involves 18 instances, ten positives and eight negatives. C4.5 algorithm is applied to X and a DT is obtained as a result. Each leave $(\mathcal{L}_i)$ of the induced DT represents one of the regions in which R^2 space has been divided. Each region is labeled with the majority class in order to detect regions with opposite label. Finally a FLD is applied to each pair of opposite regions to select a subset X_R whose elements are supposed to be support vectors. For deeper insight refer to the original work [3].

3 Methodology and Experiments

In this section, a kernel evolution process is described. Materials and experiments settings are specified.

3.1 General Overview for Kernel Evolution

Figure 3 illustrates the phases into which the kernel evolution process is split. First, a random population is created and genetically modified. Kernels obtained with genetic operators are used for training an SVM classifier. The classifier is training for a specific dataset by means of the libsvm method [15]. Accuracy is considered

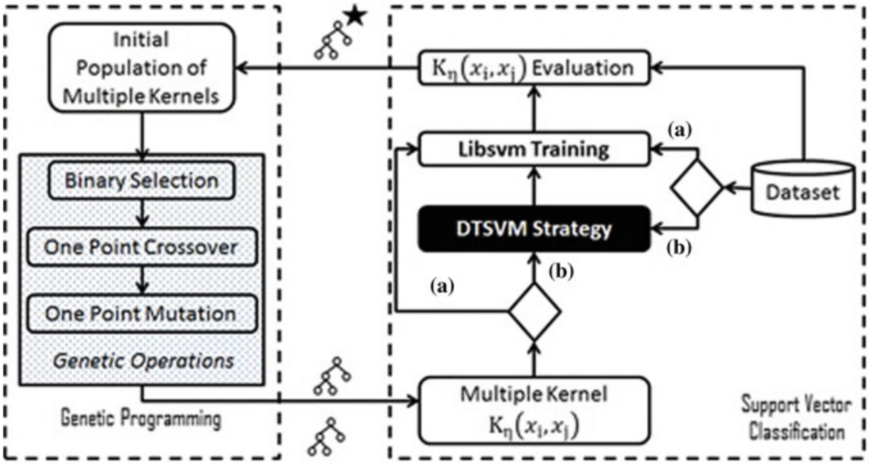


Fig. 3 Multiple kernel evolution. **a** Without data reduction. **b** Using data reduction

as a quality measure. The kernel with the best quality is passed to the population for the next generation replacing the worst individual.

In order to keep the evolution process as simple as possible, the accuracy rate for each candidate solution was calculated over the original data.

3.2 Datasets

Ten datasets with different number of instances and attributes were used in experiments. Table 2 provides relevant information about each dataset. The dataset sources are: the UCI machine learning [16], libsvm and delve [15] repositories.

Table 2 Datasets for experiments

Dataset	Instances	Attributes	Number of positives	Number of negatives	Source
Haberman	306	3	225	81	UCI
Ionosphere	351	34	124	217	UCI
Breast	683	10	444	239	UCI
Pima	768	8	500	268	UCI
Fourclass	862	2	307	555	libsvm
Splice	1000	60	517	483	delve
a4a	4781	123	1188	3593	UCI
a5a	6414	123	1284	5130	UCI
a6a	11,220	123	2692	8528	UCI
cod-rna	59,535	8	19,845	39,690	libsvm

Table 3 Configuration of algorithms for kernel evolution

Parameters for genetic programming	Value	Parameters for SVM and DTSVM	Value
Replications	50	C	2
Population size	5	d (degree)	2
Tree depth	3	σ (scale)	1
Generations	2	r (offset)	0
ϵ (Stop criterion)	$\epsilon < 10^{-3}$	DT algorithm	C4.5
Mutation rate	0.2	Impurity measure. Entropy (t)	$-\sum_{i=0}^{C_L-1} p(i\|t)\log_2 p(i\|t)$
Crossover rate	0.8		
Operator set	$\{\times, +\}$	DT pruning	Post pruning
Terminal set	{Linear, sigmoid, polynomial, RBF}	DT stop splitting criterion	10 % of dataset in one leaf
Initialization method	Grow	Proportion of support vectors candidates	10 % of closest element to the FLD threshold
Mutation method	One point		
Selection method	Binary selection		

Table 3 lists the settings for the evolution process. Replications are the times a whole evolution process was run for a specific dataset. Population size indicates the number of multiple kernels randomly generated. Tree depth specifies the maximum number of levels a tree can reach.

Generations stablishes the times the whole kernel population is genetically modified and evaluated. ϵ stands for the tolerance error in accuracy rate. C represents the SVM error penalty. d, σ and r are kernel parameters. C_L is the number of classes. $p(i|t) = \frac{|y_i = t|}{|X|}$ is the probability of example i to be in class t.

Parameters were chosen to construct a context in which the evolution process could be fully measured using a small amount of time. It is important to point out that any other parameter configuration could have been used without loss of expression since the main goal is to observe if the evolution process implementing the DTSVM method is finished faster than without implementing it under the same scenario.

All experiments were run in a computer with the next features: i7 core 3.2 GHz processor, 8 GB RAM, 100 GB SSD. Java 8 SE was the programming language and Windows 7 ultimate the operating system. The random number generator for tests was the one implemented by Random java class. For each replication the current time was considered as initial seed.

4 Results and Discussion

In Fig. 4 it is shown the time required for evolving kernels with and without using DTSVM. The vertical axis measures the time in $\log_{10}(t)$ scale, where t is in seconds. The horizontal axis measures the number of instances a dataset contains.

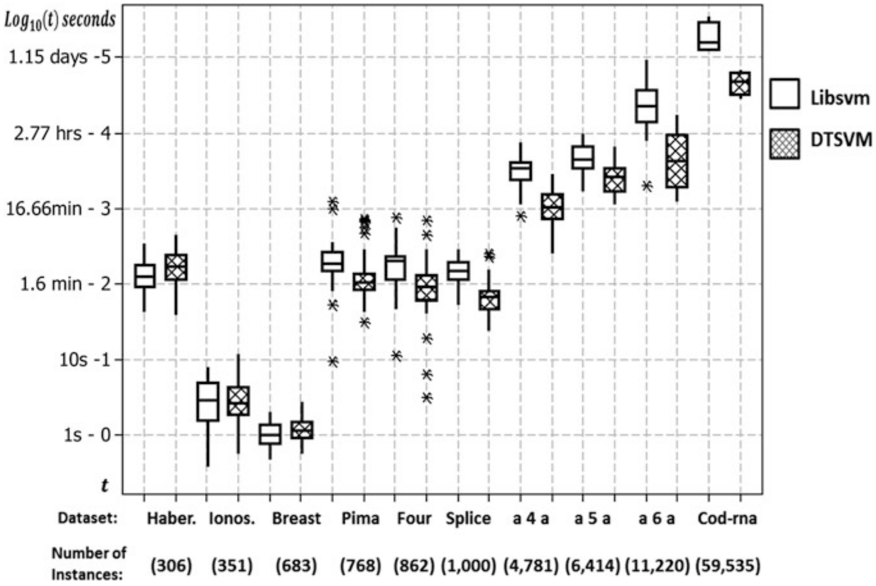


Fig. 4 Box plot of time for evolving kernels with genetic programming

Table 4 Numerical results for kernel evolution

Dataset	Instances	Method	Max.	Avg.	Med	Min	Std. dvt.
Haberman	306	libsvm	5.75	2.31	2.03	0.70	1.10
		DTSVM	7.38	3.08	2.83	0.66	1.63
Ionosphere	351	libsvm	0.13	0.05	0.05	0.01	0.03
		DTSVM	0.19	0.05	0.04	0.01	0.04
Breast	683	libsvm	0.03	0.02	0.02	0.01	0.01
		DTSVM	0.05	0.02	0.02	0.01	0.01
Pima	768	libsvm	20.48	3.83	3.11	0.16	3.33
		DTSVM	12.10	2.61	1.74	0.52	2.79
Fourclass	862	libsvm	12.83	3.28	3.28	0.19	2.05
		DTSVM	11.37	2.07	1.53	0.05	1.92
Splice	1000	libsvm	4.85	2.47	2.46	0.89	0.90
		DTSVM	4.10	1.25	1.10	0.40	0.74
a4a	4781	libsvm	124.19	55.63	56.81	13.08	22.62
		DTSVM	47.25	19.63	17.51	4.23	9.71
a5a	6414	libsvm	163.15	82.42	73.97	27.85	32.46
		DTSVM	111.58	46.87	44.01	19.03	22.53
a6a	11,220	libsvm	1570.51	486.55	386.66	33.67	355.86
		DTSVM	292.68	100.87	71.96	20.41	82.93
cod-rna	59,535	libsvm	5857.27	3321.75	2680.18	2069.38	1725.03
		DTSVM	1138.75	808.94	814.31	468.38	274.86

Times are presented in minutes

In each pair, the first box and the second box represents the behavior of training an SVM with or without using DTSVM respectively.

It can be observed from the chart that the bigger the dataset is, the more benefit from the new method is obtained. With small datasets (less than 1000 instances) the method has random behavior and the time required for evolving kernels is more likely to be dependent on the dataset complexity or any other variable rather than the dataset size.

Table 4 summarizes the statistics of training time (in minutes). For each dataset the first row represents the results of training only with libsvm. The second row shows results of applying data reduction DTSVM and then training with libsvm. Large datasets are notoriously benefited from data reduction.

5 Conclusion and Future Directions

In this chapter the DTSVM method was tested for kernel evolution on large datasets. Results indicate that evolutionary strategies could take advantage from the data reduction method when datasets are greater than 1000 instances. For smaller datasets, variations in time make explicit the need to carry out further research in order to determine if a correlation exists between the dataset size and the time required for evolving kernels with DTSVM.

Based on the fact that kernel evolution has been recently analyzed only with small datasets, and considering that methods for training SVMs faster are emerging as a current trend, authors believe that this is, likely, the first work that focus the kernel evolution for large datasets.

Results motivate to explore future directions. Authors suggest: to analyze the effect of more methods for fast training of SVM in kernel evolution; to extend the diversity of datasets to increase the confidence training time results; to study how the new method impacts in the algorithmic complexity for training an SVM.

Acknowledgments Luis Carlos Padierna García wishes to acknowledge the financial support of the Consejo Nacional de Ciencia y Tecnología (CONACYT grant 375524). The authors also thank the support of the Tecnológico Nacional de México – Instituto Tecnológico de León.

References

1. Diosan, L., Rogozan, A., Pecuchet, J.-P.: Learning SVM with complex multiple kernels evolved by genetic programming. Int. J. Artif. Intell. Tools **19**(5), 647–677 (2010)
2. Gönen, M., Alpaydin, E.: Multiple kernel learning algorithms. J. Mach. Learn. Res. **12**, 2211–2268 (2011)
3. Asdrúbal, C., Xiaoou, L., Wen, Y.: Support vector machine classification for large datasets using decision tree and fisher linear discriminant. Future Gener. Comput. Syst. **36**, 57–65 (2014)

4. Shigeo, A.: Support vector machines for pattern classification. Springer, New York (2010)
5. Essam, A.D., Hamza, T.: New empirical nonparametric kernels for support vector machines classification. Appl. Soft Comput. **13**, 1759–1765 (2013)
6. Castro, E., Gómez-Verdejo, V., Martínez-Ramón, M., Kiehl, K. A., Kalhound,V.D.: A multiple kernel learning approach to perform classification of groups from complex-valued fMRI data analysis-application to schizophrenia. NeuroImage **87**, 1–17 (2014)
7. Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., Konen, W.: Tuning and evolution of support vector machines. Evol. Intell. 1–30 (2011)
8. Padierna, L.C., Carpio, J.M., Baltazar, M.D.R., Puga, H.J., Fraire, H.J.: Muliple kernel support vector machine is np-complete. In: Gelbukh, A., Félix, C., Galicia-Haro, S. (eds.) Nature Inspired Computation and Machine Learning, Springer International Publishing, Switzerland (2014)
9. Tsang, I., Kwok, J., Cheung, P.-M.: Core vector machines-fast SVM training on very large data sets. J. Mach. Learn. Res. 363–392 (2005)
10. Vapnik, V.: Statistical Learning Theory. Wiley, New York (1998)
11. Deng, N., Tian, Y., Zhang, C.: Support Vector Machines. CRC Press, Boca Raton (2013)
12. Bollchandani, D., Sahula, V.: Exploring efficient kernel functions for support vector machines based feasibility models for analog circuits. Int. J. Des. Anal. Tools Circ. Syst. 1–8 (2011)
13. Mercer, J.: Functions of positive and negative type, and their connection with the theory of integral equations. Philoso. Trans. Roy. Soc. London A Math. Phys. Eng. Sci. **209**, 415–446 (1909)
14. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
15. Chih-Chung, C., Chih-Jen, L.: Libsvm-a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 1–27 (2011)
16. Asuncion, A., Newman, D.J.: UCI machine learning repository. University of California, Irvine, CA (2007). http://www.ics.uci.edu/~mlearn/MLRepository.html