Efficient Classification of Long Time-Series

Josif Grabocka[1], Erind Bedalli[2], and Lars Schmidt-Thieme[1]

[1] Information Systems and Machine Learning Lab
Samelsonplatz 22, 31141 Hildesheim, Germany
[2] Ph.D. candidate in University of Tirana, Lecturer in University of Elbasan, Albania
{josif,schmidt-thieme}@ismll.de,erindbedalli@gmail.com

Abstract. Time-series classification has gained wide attention within the Machine Learning community, due to its large range of applicability varying from medical diagnosis, financial markets, up to shape and trajectory classification. The current state-of-art methods applied in time-series classification rely on detecting similar instances through neighboring algorithms. Dynamic Time Warping (DTW) is a similarity measure that can identify the similarity of two time-series, through the computation of the optimal warping alignment of time point pairs, therefore DTW is immune towards patterns shifted in time or distorted in size/shape. Unfortunately the classification time complexity of computing the DTW distance of two series is quadratic, subsequently DTW based nearest neighbor classification deteriorates to quartic order of time complexity per test set. The high time complexity order causes the classification of long time series to be practically infeasible. In this study we propose a fast linear classification complexity method. Our method projects the original data to a reduced latent dimensionality using matrix factorization, while the factorization is learned efficiently via stochastic gradient descent with fast convergence rates and early stopping. The latent data dimensionality is set to be as low as the cardinality of the label variable. Finally, Support Vector Machines with polynomial kernels are applied to classify the reduced dimensionality data. Experimentations over long time series datasets from the UCR collection demonstrate the superiority of our method, which is orders of magnitude faster than baselines while being superior even in terms of classification accuracy.

Keywords: Machine Learning; Data Mining; Time Series Classification; Dimensionality Reduction

1 Introduction

Time-series classification is one of the most appealing domains of machine learning due to the abundance of application domains ranging from medicine to finance. The nearest neighbor classifier empowered with a distance metric known as Dynamic Time Warping holds the primate among accurate classifiers. However, the nearest neighbor suffers from a major drawback in terms of classification time complexity which deteriorates to quartic time for the whole dataset. In order to overcome the scalability problems associated with nearest neighbor we propose a fast matrix factorization in order to reduce the data dimensionality and then Support Vector Machines for classification. We show through experimental results that the both the accuracy and the classification time is significantly improved, while the training time is competitive to a standard Support Vector Machines training duration.

2 Related Work

2.1 Time-Series Classification

The recent decades have witnessed a plethora of methodologies addressed at the classification of time series. The methodologies vary from Neural Networks [1], Bayesian Networks [2] to SVMs [3].

Dynamic Time Warping Among the most successful methods proposed within the scope of time-series classification are distance based similarity metrics. Various distance metrics have been proposed, however the most widely recognized and accurate metric is the so-called Dynamic Time Warping (DTW) [4]. Dynamic Time Warping is able to detect distortions between series of the same class via computing the minimum warping alignment path of the respective time points. The minimum time points alignment is computed through a dynamic algorithm, which constructs a cost matrix where each cell represent the cumulative distance for the partial alignment up to the indexes of its coordinates [5, 6]. DTW is used in combination with the nearest neighbor classifier, denoted DTW-NN. Recent studies have pointed out that DTW-NN is a hard-to-beat baseline in terms of classification accuracy [7]. In order to speed up the classification time of DTW, a warping window concept has been introduced in order to reduce the number of computations of the cost matrix cells, by omitting candidate warping paths having deviations from the matrix diagonal, i.e from the euclidean alignment of time points [6]. In this study we are comparing our method against the DTW-NN.

2.2 Matrix Factorization

Matrix factorization is a variant of dimensionality reduction that projects data into a reduced/latent/hidden data space which usually consists of lower dimensions than the original space [8, 9]. Different approaches have been unified under a generalized Bregman divergence theory [10]. Matrix factorization has been applied in domains involving time-series data as in music transcription [11], up to EEG processing [12] In comparison we are going to use very fast variations of matrix factorization with very low dimensions, fast learning rate and early stopping.

2.3 Support Vector Machines (SVM) Classification

Support Vector Machines (SVM) are considered to be one of the best off-the-shelf classifier for a wide application domains of machine learning. The success of support vector machines is based on their principle to find the maximum margin decision boundary hyperplane that accurately splits class regions [13]. The training process of SVMs is done by optimizing the maximum margin objective, usually in the dual representation of the objective function. In order to overcome problems with non-linear separability of certain datasets, introduction of slack variables has been applied to allow regularized disobedience from the decision boundary. In addition kernel theory has been combined with the dual learning of SVMs in order to offer various types of non-linear expressiveness to the decision boundary [14]. We are applying SVMs for classifying the projected data in the latent space.

3 Motivation

The time complexity of classifying long time series is obviously determined by the runtime complexity of the method performing such classification, while the complexities of those methods are functions of series lengths. In this section we provide more insight into the diagnosis and the cure for time complexity issues of lengthy series. However, before starting the analysis we need to provide a brief description of DTW-NN.

3.1 Dynamic Time Warping and Nearest Neighbor

A warping path between two series $A = (A_1, ..., A_n)$ and $B = (B_1, ..., B_m)$, denoted as $\pi^{A,B}$ is defined as an alignment $\tau^{A,B} = (\tau_1^{A,B}, \tau_2^{A,B})$ between the elements of A and B. The alignment starts and ends with extreme points,

$$P = |\tau^{A,B}|$$
$$1 = \tau^{A,B}(1)_1 \leq ... \leq \tau^{A,B}(P)_1 = n$$
$$1 = \tau^{A,B}(1)_2 \leq ... \leq \tau^{A,B}(P)_2 = m$$

while involving incremental alignment of adjacent pairs as:

$$\begin{pmatrix} \tau^{A,B}(i+1)_1 - \tau^{A,B}(i)_1 \\ \tau^{A,B}(i+1)_2 - \tau^{A,B}(i)_2 \end{pmatrix} \in \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}$$

The overall distance of the points aligned by a warping path is computed as the sum of distances of each aligned pair. Such distance is called the Dynamic Time Warping distance [6].

$$\text{DTW}(A, B) = \underset{\tau^{A,B}}{\text{argmin}} \sum_{p=1}^{|\tau^{A,B}|} \left(A_{\tau^{A,B}(p)_1} - B_{\tau^{A,B}(p)_2} \right)^2$$

Dynamic Time Warping distance is practically computed by a dynamic algorithm, which is computed via a cost matrix, denoted W. Each cell of the cost matrix is computed as follows:

$$\begin{aligned}\text{DTW}(A,B) &= \mathbf{W}_{\text{length}(\mathbf{A}),\text{length}(\mathbf{B})}\\ \mathbf{W}_{1,1} &= (A_1 - B_1)^2\\ \mathbf{W}_{i,j} &= (A_i - B_j)^2 + \min(\mathbf{W}_{i-1,j}, \mathbf{W}_{i,j-1}, \mathbf{W}_{i-1,j-1}) \qquad (1)\end{aligned}$$

The nearest neighbor classifier based on DTW distance metric is described in Algorithm 1 and basically predicts the label of a test instance as the label of the closest train instance.

Algorithm 1 DTW-NN

```
Require: Training set $D$, Test instance $I$
Ensure: Predicted label of $I$
  nearestNeighbor ← $\text{argmin}_{I' \in D}$ DTW($I, I'$)
  return nearestNeighbor.label
```

3.2 The *Curses* of Dimensionality and Complexity

The computation of the DTW distance from Equation 1 between two series A and B requires the computation in total of (length(A) $\times$ length(B)) cells which make the distance metric an $O(n^2)$ operation. In order to classify a series instance using the nearest neighbor classifier from Algorithm 1, we will have to compare its distance against all training set instances which requires $O(n)$ calls to DTW computation. Therefore the overall classification time complexity of classifying an instance is $O(n^3)$, while the classification of a whole test set becomes $O(n^4)$. Cubic and/or quadric time complexities creates prohibitive possibilities for low computational devices, or systems where the response time is critical. Please note that especially in long time series the DTW computation tend to become expensive, therefore we can metaphorically call this behavior as *the curse of complexity.*

SVMs are much faster in terms of classification speed, however a learning phase step is required first, because a maximum margin decision boundary has to be created. In contrast, the nearest neighbor classifier requires no training step at all. Even worse the classification and learning phases of SVMs can be time consuming in case the number of features (here series length) is large. Such phenomenon is known as *curse of dimensionality.* The purpose of this study is to make the classification time of SVMs faster through first projection the data into a latent space having much less features with even better accuracy. This speedup comes with the additive learning time cost of the projection method, therefore we will propose a very fast projection technique. The overall learning and classification time of our method is shown to be much faster than DTW-NN. In the end of Section 4.3 we explain that our method has an $O(n)$ classification time complexity.

4 Proposed Method

Throughout this study we propose e method that aims at solving the problem of time-series classification as formalized in section 4.1. Our method applies a Fast Matrix Factorization, described in section 4.2, in order to project the data to a lower dimensionality. Finally SVMs classifier, section 4.3, is applied to the projected data for classification.

4.1 Problem Description

Given a training time series dataset $X_{train} \in \mathbb{R}^{N \times M}$ consisting of N time series each of M points length and observed target variables $Y_{train} \in \mathbb{N}$, then the task is to predict the labels Y_{test} of a given test set of series $X_{test} \in \mathbb{R}^{N' \times N}$.

4.2 Fast Matrix Factorization

Matrix Factorization is a technique to decompose a matrix as the dot product of other matrices having typically lower dimensionality. In our study, the time-series dataset $X \in \mathbb{R}^{(N+N') \times M}$ will be approximated by the dot product of the projected latent data $\Phi \in \mathbb{R}^{(N+N') \times K}$ and the matrix $\Psi \in \mathbb{R}^{K \times M}$ as in Equation 2. The value of K determine the dimensionality of the projected space.

$$X \approx \Phi \cdot \Psi \tag{2}$$

Such an approximation is converted to an objective function that can be written in terms of a minimization objective function, denoted L, as in Equation 3. The loss terms are euclidean distances and can be expanded as in Equation 4. The loss terms on the right preceded by λ coefficients are regularization terms which prevent the over-fitting of Φ and Ψ.

$$\underset{\Phi,\Psi}{\operatorname{argmin}}\ L(X,\Phi,\Psi) = ||X - \Phi \cdot \Psi||^2 + \lambda_\Psi ||\Psi||^2 + \lambda_\Phi ||\Phi||^2 \tag{3}$$

$$\begin{aligned}\underset{\Phi,\Psi}{\operatorname{argmin}}\ L(X,\Phi,\Psi) &= \sum_{i=1}^{N+N'} \sum_{j=1}^{M} \left(X_{i,j} - \sum_{k=1}^{K} \Phi_{i,k} \Psi_{k,j} \right)^2 \\ &\quad + \lambda_\Phi \sum_{i=1}^{N+N'} \sum_{k=1}^{K} \Phi_{i,k}^2 + \lambda_\Psi \sum_{k=1}^{K} \sum_{j=1}^{M} \Psi_{k,j}^2 \end{aligned} \tag{4}$$

The solution of the objective function is carried through stochastic gradient descent where we randomly correct the loss created by each cell $X_{i,j}$. The corresponding partial derivatives can be derived as follows:

$$\text{Let: } e_{i,j} = X_{i,j} - \sum_{k=1}^{K} \Phi_{i,k} \Psi_{k,j} \tag{5}$$

$$\frac{\partial L_{X_{i,j}}}{\partial \Phi_{i,k}} = -2 e_{i,j} \Phi_{i,k} + 2 \lambda_\Phi \Phi_{i,k} \tag{6}$$

$$\frac{\partial L_{X_{i,j}}}{\partial \Psi_{k,j}} = -2 e_{i,j} \Psi_{k,j} + 2 \lambda_\Psi \Psi_{k,j} \tag{7}$$

In order to obtain final versions of the latent matrices Φ, Ψ we learn them through a stochastic gradient descent learning as shown in Algorithm 2. The algorithm iterates through every cell $X_{i,j}$ and updates the cells of the latent matrices until the loss is minimized. In the end of the learning the latent matrix Φ is fed to the classifier.

Algorithm 2 Stochastic Gradient Descent Learning

```
Input: Time-series dataset $X$, Learning rate $\eta$, Maximum Iterations $MaxIterations$
Output: $\Phi, \Psi$
 1: Initialize $\Phi, \Psi$ randomly
 2: $previousLoss \leftarrow MaxValue$
 3: $currentLoss \leftarrow L(X, \Phi, \Psi)$
 4: $numIterations \leftarrow 0$
 5: while $currentLoss < previousLoss \wedge numIterations \leq MaxIterations$ do
 6:   for $i = 1$ to $N + N'$ do
 7:     for $j = 1$ to $M$ do
 8:       for $k = 1$ to $K$ do
 9:         $\Phi_{i,k} \leftarrow \Phi_{i,k} - \eta \cdot \frac{\partial L_{X_{i,j}}}{\partial \Phi_{k,j}}$
10:         $\Psi_{k,j} \leftarrow \Psi_{k,j} - \eta \cdot \frac{\partial L_{X_{i,j}}}{\partial \Psi_{k,j}}$
11:       end for
12:     end for
13:   end for
14:   $previousLoss \leftarrow currentLoss$
15:   $currentLoss \leftarrow L(X, \Phi, \Psi)$
16:   $numIterations \leftarrow numIterations + 1$
17: end while
18: return $\Phi, \Psi$
```

In order to speed up the factorization there are three main steps that can be taken. The latent dimensionality, parameter K specifying latent dimensionality of matrices Φ, Ψ, is selected to be as low as $K = c \times cardinality(Y)$, meaning a small multiple c of the number of labels. In addition the learning rate η is set to be large, which forces the optimization to quickly converge towards the global minimum of the quadratic objective function in Equation 4. The final element that speeds the convergence relies on stopping the iterations via a limited maximum iterations count.

4.3 Support Vector Machines

The Support Vector Machines, hereafter denoted as SVMs, are a classifier that aims at finding the maximum margin separating decision boundary among class regions [15]. The decision boundary lies in the form of a hyperplane, denoted **w**. For binary classification the target variable y is binary $y \in \{-1, +1\}$. The classification of a test instance is computed the sign of the dot product between the hyperplane and the instance vector, as in Equation 8.

$$\hat{y}_{test} = \text{sign}(\langle w, x_{test} \rangle + w_0) \tag{8}$$

The maximum margin hyperplane is computed by solving the optimization function in Equation 9. Such formulation is known as the soft-margin primal form [13].

$$\begin{aligned}\text{minimize} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \\ & \text{subject to:} \\ & y_i(\langle w, x_i\rangle + w_0) \geq 1 - \xi_i, \text{and } \xi_i \geq 0, \;\; i = 1, ..., n\end{aligned}$$

The so called slack variables, defined in Equation 9 represent the violation of each series instance from the boundary with the objective aim of minimizing the violations.

$$\xi_i = \max(0, 1 - y_i(\langle w, x_i\rangle + w_0)) \tag{9}$$

The primal form objective function is manipulated by expressing the inequality conditions via Lagrange multipliers denoted α_i, one per instance. Then the objective function is solved for w and w_0 and the dual form is yield as shown in Equation 10.

$$\begin{aligned}\max \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K(x_i, x_j) \\ \text{subject to:} \;\; & 0 \leq \alpha_i \leq C, \;\; i = 1, ..., n \\ & \sum_{i=1}^{n}\alpha_i y_i = 0\end{aligned} \tag{10}$$

In order to classify datasets exhibiting non-linear separation, the dot product present in the dual objective function, $\langle x_i, x_j\rangle$, is substituted by the so called kernel trick which is shown as $K(x_i, x_j)$ [14]. A typical kernel, the inhomogeneous polynomial one, is presented in Equation 11, where d is the degree of the polynomial and c a constant.

$$\langle x_i, x_j\rangle \rightarrow K(x_i, x_j) = (\langle x_i, x_j\rangle + c)^d \tag{11}$$

The purpose of the kernel trick approach is to express the feature space in terms of a higher dimensionality space, such that a linear decision boundary in the high dimensional space would represent a non-linear boundary in the original one. The solution of the dual problem is carried through dedicated optimization algorithms. The ultimate decision boundary can be formed as a function of the solved Lagrange multipliers as depicted by Equation 12. Please note that the non-zero α_j coefficients correspond to the so-called support vector instances (x_j, y_j).

$$y_i = \text{sign}(\sum_{j=1}^{n}\alpha_j y_j K(x_j, x_i)) \tag{12}$$

Once the decision boundary is learn then a new instance can be classified as a summation iteration over the support vectors, which make the operation of magnitude $O(a \times b)$, where a is the number of support vectors and b the complexity of the dot product of two instance's features. Please note that a is a fraction of the number of instances, so still $O(n)$ at worst-case, while b is linearly proportional to the number of features. So SVM classification time deteriorates to $O(n^2)$ in worst case scenario. In our method we will project the data into a very low dimensionality, so b will be quasi $O(1)$ and aggregatively our method will have a worst-case classification time complexity of $O(n)$.

5 Experimental Setup

The analysis and the experiments served to analyze the proposed methods are tested on the longest five datasets of the UCR time-series data collection [3]. The statistics of the selected datasets are found in Table 1.

Table 1. Statistics of Datasets

Dataset	Number of Instances	Series Length	Number of Labels
CinC_ECG_Torso	1420	1639	4
Haptics	462	1092	5
InlineSkate	650	1882	7
Mallat	2400	1024	8
StarLightCurves	9236	1024	3

In order to test our method, which we denote as Fast Matrix Factorization and SVM, hereafter denoted as FMF-SVM, we run experiments against the following implemented baselines:

– **DTW-NN:** The nearest neighbor with Dynamic Time Warping is a hard to beat classifier in the time-series domain.
– **E-NN:** The nearest neighbor with Euclidean Distance is a fast version classifier compared to DTW-NN.
– **SVM:** Support Vector Machines are a strong off-the-shelf classifier.

The hyper-parameters of our method are searched via grid search using only the validation set in a 5-cross validation fashion. The values of the learning rate η are searched from a range of $\{10^{-1}, 10^{-2}, 10^{-3}\}$, the latent dimensionality among $\{1, 2, 3, 4\} \times \mathit{cardinality}(Y)$, parameter C of SVMs is selected from $\{10^{-1}, 1, 10^{1}\}$, the maximum iterations from $\{100, 200, 300\}$, and finally the degree of the polynomial kernel is chosen one of $\{2, 3, 4\}$. The combination yielding the minimum error on the validation split, is tested over the test split.

6 Results

Our proposed method FMF-SVM exhibits excellent classification accuracy by producing the smallest misclassification rates compared to the baselines in all the datasets, as demonstrated in Table 2.

[3] `www.cs.ucr.edu/~eamonn/time_series_data`

Table 2. Misclassification Rate Results Table ***(5-fold cross validation)***

Dataset	**FMF-SVM**		**DTW-NN**		**E-NN**		**SVM**	
	mean	*st.dev.*	*mean*	*st.dev.*	*mean*	*st.dev.*	*mean*	*st.dev.*
CinC_ECG_Torso	**0.0007**	0.0016	**0.0007**	0.0001	0.0014	0.0020	**0.0007**	0.0035
Haptics	**0.4903**	0.0118	0.5484	0.0025	0.5745	0.0750	0.5162	0.0405
InlineSkate	**0.5098**	0.0151	0.5131	0.0018	0.5603	0.0297	0.5197	0.0638
Mallat	**0.0150**	0.0027	0.0162	0.0001	0.0163	0.0063	0.0196	0.0084
StarLightCurves	**0.0633**	0.0074	0.0652	0.0001	0.1139	0.0044	0.0933	0.0449

The proposed method requires an additional matrix factorization step which elongates the overall training time of building the model. Nevertheless the learning time is not prohibitive, and it is practically feasible as Table 3 shows.

Table 3. Learning Times (seconds)

Dataset	**FMF-SVM**		**SVM**	
	mean	*st.dev.*	*mean*	*st.dev.*
CinC_ECG_Torso	2.44	0.17	10.74	1.14
Haptics	223.55	3.86	2.80	0.11
InlineSkate	214.74	5.17	10.85	0.85
Mallat	840.81	30.42	10.69	0.32
StarLightCurves	2354.30	568.88	1033.51	656.81

Finally, in addition to being superior in classification, our method is also extremely faster in terms of classification time. As Table 4 proves, FMF-SVM is by far superior in terms of classification time regarding new test instances. The presented results support our theoretic complexity analysis of Section 3.2 and Section 4.3. (Note that $M = 10^6$).

7 Conclusion

Throughout this study we presented a new approach addressing the problem of classifying long time series. In comparison to state-of-art similarity based nearest neighbor classifiers, which deteriorate up to cubic orders of classification

Table 4. Classification Times Results (milliseconds)

Dataset	**FMF-SVM**		**DTW-NN**		**E-NN**		**SVM**	
	mean	*st.dev.*	*mean*	*st.dev.*	*mean*	*st.dev.*	*mean*	*st.dev.*
CinC_ECG_Torso	**0.21**	0.01	182900.92	3403.49	72.22	1.36	11.49	2.45
Haptics	**0.68**	0.03	24985.02	116.01	16.64	0.48	3.48	0.08
InlineSkate	**0.73**	0.04	98908.95	354.88	34.92	0.41	11.09	0.27
Mallat	**0.64**	0.02	115535.66	572.01	75.61	1.52	9.05	0.31
StarLightCurves	**0.57**	0.05	256.87M	4.54M	286.76	9.09	15.99	4.76

time complexity, we propose a fast data projection to low dimensions and then a SVMs classification on the latent space. Overall, our method's classification time complexity is only $O(n)$ at worst-case scenario. Experiments over long time-series datasets demonstrate that our method is clearly, both the fastest, and the most accurate compared to selected state-of-art baselines.

8 Acknowledgement

Funded by the Seventh Framework Programme of the European Commission, through project REDUCTION (# 288254). `www.reduction-project.eu`

References

1. Kehagias, A., Petridis, V.: Predictive modular neural networks for time series classification. Neural Networks **10**(1) (1997) 31–49
2. Pavlovic, V., Frey, B.J., Huang, T.S.: Time-series classification using mixed-state dynamic bayesian networks. In: CVPR, IEEE Computer Society (1999) 2609–
3. Eads, D., Hill, D., Davis, S., Perkins, S., Ma, J., Porter, R., Theiler, J.: Genetic Algorithms and Support Vector Machines for Time Series Classification. In: Proc. SPIE 4787; Fifth Conference on the Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation; Signal Processing Section; Annual Meeting of SPIE. (2002)
4. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.J.: Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB **1**(2) (2008) 1542–1552
5. Berndt, D.J., Clifford, J.: Finding patterns in time series: A dynamic programming approach. In: Advances in Knowledge Discovery and Data Mining. (1996) 229–248
6. Keogh, E.J., Pazzani, M.J.: Scaling up dynamic time warping for datamining applications. In: KDD. (2000) 285–289
7. Keogh, E.J., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowl. Inf. Syst. **7**(3) (2005) 358–386
8. Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. IEEE Computer **42**(8) (2009) 30–37
9. Srebro, N., Rennie, J.D.M., Jaakola, T.S.: Maximum-margin matrix factorization. In: Advances in Neural Information Processing Systems 17, MIT Press (2005) 1329–1336
10. Singh, A.P., Gordon, G.J.: A unified view of matrix factorization models. In: ECML/PKDD (2). (2008) 358–373
11. Smaragdis, P., Brown, J.C.: Non-negative matrix factorization for polyphonic music transcription. In: In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics. (2003) 177–180
12. Rutkowski, T.M., Zdunek, R., Cichocki, A.: Multichannel EEG brain activity pattern analysis in time-frequency domain with nonnegative matrix factorization support. International Congress Series **1301** (2007) 266–269
13. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2010)
14. Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, USA (2001)
15. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**(3) (1995) 273–297