Efficient Sampling and Handling of Variance in Tuning Data Mining Models

Patrick Koch and Wolfgang Konen*

Department of Computer Science, Cologne University of Applied Sciences,
51643 Gummersbach, Germany,
{patrick.koch, wolfgang.konen}@fh-koeln.de

Abstract. Computational Intelligence (CI) provides good and robust working solutions for global optimization. CI is especially suited for solving difficult tasks in parameter optimization when the fitness function is noisy. Such situations and fitness landscapes frequently arise in real-world applications like Data Mining (DM). Unfortunately, parameter tuning in DM is computationally expensive and CI-based methods often require lots of function evaluations until they finally converge in good solutions. Earlier studies have shown that surrogate models can lead to a decrease of real function evaluations. However, each function evaluation remains time-consuming. In this paper we investigate if and how the fitness landscape of the parameter space changes, when only fewer observations are used for the model trainings during tuning. A representative study on seven DM tasks shows that the results are nevertheless competitive. On all these tasks, a fraction of 10-15% of the training data is sufficient. With this the computation time can be reduced by a factor of 6-10.

Keywords: Machine learning, parameter tuning, sampling, SVM, sequential parameter optimization.

1 Introduction

Data Mining (DM) is an interesting field for applying Computational Intelligence (CI) techniques. Although CI methods can generate good and robust solutions for global optimization problems, it is known that they sometimes require a large number of function evaluations. Unfortunately, data mining tasks are usually very expensive to evaluate and quick solutions are requested by the users. In this paper we investigate how computation time can be saved in order to make CI methods more applicative for DM tasks.

We claim the following hypotheses:

H1 Tuning results are more subject to noise when smaller fractions X of the training data are used.

* This work has been partially supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grant SOMA ("Ingenieurnachwuchs" 2009).

H2 Tuning with smaller fractions X will usually lead to increased prediction errors, but the optimal design points found by a robust optimizer will nevertheless be competitive (as long as X is not *too* small).

If we rewrite **H1** a little, we can come to the conclusion that the variance should be higher when the training set size is smaller. If **H1** holds, we should be able to measure this effect directly by an increased variance of the model error. If we can confirm **H2**, considerable computation speedups in tuning DM models are possible.

Previous work in analyzing the effects of the chosen sample size has been mainly done in fields like statistics and machine learning. A good overview about different strategies to sample data can be found in Cochran [3]. A description of resampling methods for optimization in data mining has been given by Bischl *et al.* [2]. Raudis and Jain [14] and Jain and Zongker [7] discuss the influence of sample sizes on the results of feature selection, which is in a certain way related to the parameter optimization task in this article. In statistics, sample sizes are frequently discussed in terms of statistical studies like significance tests [11]. In machine learning, sampling strategies like the bootstrap [5] have been well analyzed and are often applied in practice. However, to our knowledge no study exists where the size of the underlying training data is diminished during parameter tuning.

2 Methods

2.1 Learning Algorithms

As a learning algorithm we experimented with the Support Vector Machine (SVM) [16], since SVM is known to be very sensitive to its parameter settings. We used the *libsvm* implementation in `R` from the *e1071* package[1]. However, all experiments can be also performed with any other (supervised) machine learning algorithm. Here, we restricted ourselves to SVM, since it appeared to be best suited for our experiments.

SVM needs several hyperparameters to be set to work properly. First of all it requires a kernel function to allow for non-linear class boundaries. The choice of the kernel function is a crucial decision in machine learning and must be considered carefully. However, some kernel functions are good-working for several problems. For instance, the radial basis function (RBF) kernel belongs to the most popular kernel functions and defines the similarity of inputs x and z by

$$k(x, z) = exp(-\gamma(||x - z||)) \tag{1}$$

For our needs the RBF kernel function is well-suited, since it comes along with the hyperparameter γ which has to be set anew for each data. Small values of γ indicate large influences of given data points, whereas large γ values mean that the influence of the data points is more restricted. Besides the γ kernel

[1] Software available from http://cran.r-project.org/web/packages/e1071/index.html

parameter SVM regularizes points which do not lie in the hyperplane fitted by the algorithm. Therefore it uses a so-called regularization parameter C (for cost), which is important for finding a good balance of the underlying optimization problem of the SVM and correct classification of training examples. For more details on SVM we refer the interested reader to the literature [15].

2.2 Tuning Algorithms

Parameter tuning tasks in machine learning can be modelled as noisy single-objective optimization problems. Any parameter setting of the learning parameters is called a *design*. For each design we can measure the quality by training a model (running the learning algorithm using some training data) and applying the trained model to new test data. Note that although the underlying learning algorithm may be deterministic, the tuning task can be nevertheless stochastic, because the training and test samples of the data are drawn at random. Hence, the robustness and generalization ability of the optimization algorithm is an important criterion for the tuning task. Konen *et al.* [10] therefore compared tuning algorithms like the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [6] with the Sequential Parameter Optimization Toolbox (SPOT) [1]. SPOT is a heuristic which employs surrogate models of the objective function. The advantage of such strategies is that the number of real function evaluations can be reduced, since parts of the optimization can be performed on the surrogate model. In a comparative study [10], SPOT performed best under strongly limited budgets. It was noted that CMA-ES can give similar results, presuming that enough function evaluations are allowed. Other statistical methods like Latin hypercube sampling (LHD) [12] uniformly distribute design points over the search space: their performance and accuracy usually diminishes for larger search space dimensions.

2.3 Objective Function

We perform a tuning of all parameters which are relevant for our machine learning task. Any single-objective optimizer requires an objective function, in order to evaluate the quality of the parameter designs. In our case we distinguish between

(a) the model error on validation data during tuning, and
(b) the error obtained with the best parameters from tuning on independent test data (data which has not been used throughout the whole tuning process)

During tuning the objective function value is the fraction of wrongly classified pattern in the validation set. The unbiased estimator for the model's error on new data is th fraction of wrongly classified pattern in the test set.

3 Experimental Setup

Parameter optimization in machine learning can be challenging: the training time of SVM often grows quadratically with the number of training patterns

[13]. Since this might be one reason why parameter optimization is seldom done in practice, we try to find new ways to make tuning of DM tasks less costly: We vary the number of training patterns used during parameter tuning. In our optimization task the most time consuming part is the evaluation of the objective function, because in each evaluation a complete SVM training is performed (the runtime of optimizers like SPOT can be also expensive, but are neglectable in our case, since we only use fast surrogate models and small designs). We investigate how much time can be saved by reducing the training data used for parameter tuning and if parameters with small training fractions are competitive with parameters tuned on the total training set.

3.1 Datasets

All experiments presented in this study were performed using datasets from the UCI website,[2] which can be regarded as the simpler datasets, the DMC 2007 data from the data mining cup 2007, and a real-world dataset from water ressource management (AppAcid). In Tab. 1 an overview about the datasets and their sizes is given. We present the minimal training set size for each dataset, and also the sizes of the test and validation sets. Each model is evaluated a) during tuning on the validation data and b) after tuning on the independent test data. The sizes for the test and validation sets are equal and stay constant all the time at 20% of the total dataset size. In Tab. 1 these sizes are given as *Validation and Test Size*.

Table 1. Datasets used for the training set size experiments.

Dataset	Records	Min. Training Size	Max. Training Size	Validation and Test Size	Number of Parameters
Sonar	208	8	124	41	2
Glass	214	8	128	42	2
Liver	345	13	207	69	2
Ionosphere	351	14	210	70	2
Pima	768	30	460	153	2
AppAcid	4400	176	2640	880	12
DMC-2007	50000	2000	30000	10000	7

Every time a fixed set of 20% of the patterns was set aside prior to tuning for testing purposes. From the remaining 80% of the data (subset D_{train}), we use a fraction X from $X_{min} = 5\%$ to $X_{max} = 75\%$ for training and a fraction of 25% for validation (which is 20% of all data). See Tab. 2 for illustration. At the end of tuning a best design point is returned. Using this best design point, we ran a final 'full' training with all data in D_{train} (80%) and evaluated the trained

[2] http://archive.ics.uci.edu/ml/

model on the test set (20%). Since the training, validation and test sets were drawn at random we repeated all of our experiments ten times.

Table 2. Splitting of data. We use fractions from 4% to 60% of the data for model training, 20% for validation during the tuning and the remaining 20% for independent testing.

Training $\longleftrightarrow$	not used	Validation	Test

While a first benchmark of tuning algorithms has been performed by Konen *et al.* [10], it remained unclear, if the results also hold for smaller training set sizes. Now we also compare SPOT as a tuning algorithm with LHD as a simple, but robust sampling heuristic. LHD is based on the following procedure: We chose random and roughly equally distributed design points from the region of interest and evaluated the design 3 times. Again, the best point is taken for the 'full' training as above.

3.2 SPOT Setup

SPOT can be controlled through various settings, which have to be adapted slightly for each task. We briefly present our settings for the experimental study here (Tab. 3). With 150 function evaluations for the UCI experiments we chose a rather large number of evaluations compared to the other (real-world) datasets. Our aim was to achieve a good and clear convergence to an optimum. Out of this reason we considered to analyze simpler datasets first, since complex datasets require much more time for such experiments. Nevertheless we also set a number of 200 function evaluations for AppAcid, which proved to be a good and sufficient number in one of our last studies [10]. It has to be noted that the dimensionality of the parameter space for AppAcid is 12, while it is only 2 for the UCI benchmarks.

As region of interest (ROI) for the UCI datasets we set quite large ranges (as we have enough evaluations for these benchmarks). We vary the range of γ between $[0.0, 1.0]$ and the range of cost C between $[0.0, 10.0]$. For the other applications (AppAcid, DMC2007) we relied on the same settings as in our previous experiments, see [10] for more information.

3.3 Sampling Strategies

First of all, k subsets $D_1, ..., D_k \subset D_{train}$ of the complete training data D_{train} lead to models $M_1, .., M_k$ which are presumably not equal when the training data subsets are not equal, although hyperparameters γ and C are identical (on the same training data SVM is deterministic). Thus we have $D_1 \neq D_2 \neq ... \neq D_k \rightarrow M_1 \neq M_2 \neq ... \neq M_k$. Different strategies are possible for sampling the training subsets during tuning:

Table 3. SPOT Configuration

Setting	UCI	AppAcid
function evaluations	150	200
initial design size	10	24
sequential design points	3	3
sequential design repeats	$\{1, 3\}$	2

(A) Choose a subset $D_1 \subset D_{train}$ *once* and use it for the whole parameter optimization process. We call this option *parameter tuning without resampling.*
(B) In each call i of the objective function, choose a new subset $D_i \subset D_{train}$ for training. If a design point is evaluated repeatedly, e. g. n times, we choose different subsets $D_1, D_2, ..., D_n$, train models for them and use an aggregation function (here: the mean) to return an objective function value. We call this option *parameter tuning with resampling.*

4 Results

4.1 Tuning Results

Exemplarily we present boxplots of the SPOT hyperparameter tuning for the Sonar and AppAcid datasets in Fig. 1.[3] We distinguish between the error achieved on the validation data when using only a smaller training set fraction X (*Error VALI Set*) and the independent test set error when a complete re-training with the optimal parameter setting is performed (*Error TEST Set*). All results in Fig. 1 were optimized using SPOT with sampling strategy (B) and a total number of 3 repeated evaluations for each design point. The validation error in both plots is clearly increasing if the training set size X is reduced from $X = 75\%$ to $X = 5\%$. However, the same does not necessarily hold for the test data. While the mean errors and their variances are large for small training fractions, they are roughly constant and small for all $X \geq 15\%$.

This is a promising result, as it may allow us to use only a subsample of the complete data during tuning. As we can see in Tab. 4, good tuning results with a very small number of training data (here X=10%) were observed as well for all other datasets. If we take the best parameters from such a tuning process and re-train *once* with the complete training data, we obtain results on independent test data which are competitive with a much more time-consuming 'full' tuning.

We observed on all considered datasets, that the prediction accuracies of single models on different data are very noisy. Thus, the chosen sampling strategy (see Sec. 3.3) has an impact on the final results. A comparison of the sampling strategies (A) and (B) showed that we can achieve the most stable results using strategy (B) and a number of repeated evaluations greater than 1 (from now on we always set the repeats to 3 in our study). We think that repeated evaluations

[3] A complete set of all boxplot results and accompanying material is available for the interested reader from http://gociop.de/about/people/koch/ppsn2012/.

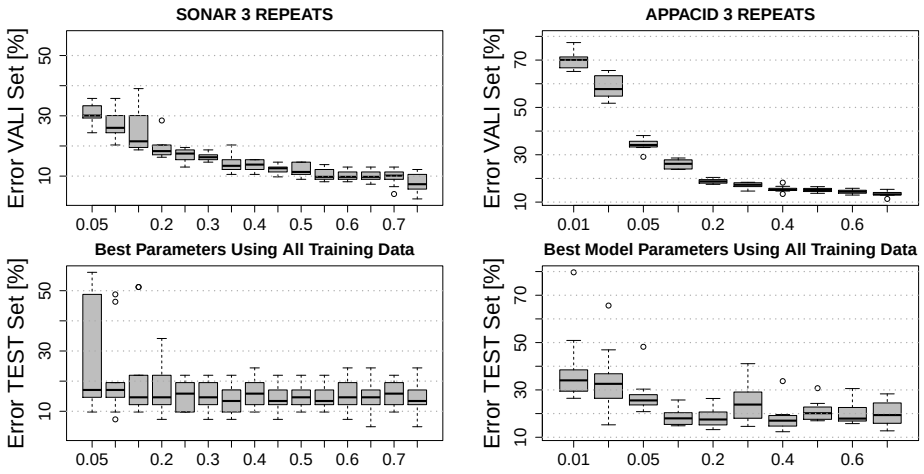

Fig. 1. Tuning results with SPOT for Sonar and AppAcid datasets using 3 repeats. The x-axis shows the training set fraction X.

Table 4. Test-set error rates (mean and standard deviation of ten runs) for all datasets investigated. We show results when using small and full training data during tuning, after retraining once with best parameters found. In case of DMC-2007 we show relative gain instead of error rate.

Dataset	X=10% median (std.dev.)	X=75% median (std.dev.)
Sonar	17.07 (14.3)	13.41 (5.5)
Glass	28.57 (7.7)	28.57 (7.6)
Liver	36.23 (4.3)	31.88 (6.5)
Ionosphere	5.71 (2.4)	5.71 (2.4)
Pima	23.20 (2.5)	23.20 (3.9)
AppAcid	17.99 (3.3)	19.38 (5.5)
DMC-2007	14.73 (2.0)	15.66 (1.0)

are an important factor to circumvent wrong decisions of the optimizer operating in a noisy environment.

Regarding the comparison of SPOT and LHD we show in Fig. 2 that SPOT usually yields in better parameter settings when trained with a fraction of the data, especially for $X \leq 10\%$. Results degrade for very low training set sizes (1% or 2%). They are however competitive for all $X \geq 10\%$ (SPOT) and $X \geq 20\%$ (LHD): The tuning results are as good as with a tuning on the 'full' training data and the models generalize well on unseen data.

4.2 Landscape Analysis

Fig. 3 shows a comparison of the surrogate surfaces for a small (X=10%) and large training set size on the Sonar dataset. We used a surrogate model based on Gaussian Processes (GP) [8]. When only few training data were used (left plots), the SPOT and LHD landscapes are both relatively flat, with shallow minima near $\gamma \approx 0$. These minima are however a good indicator for the minima

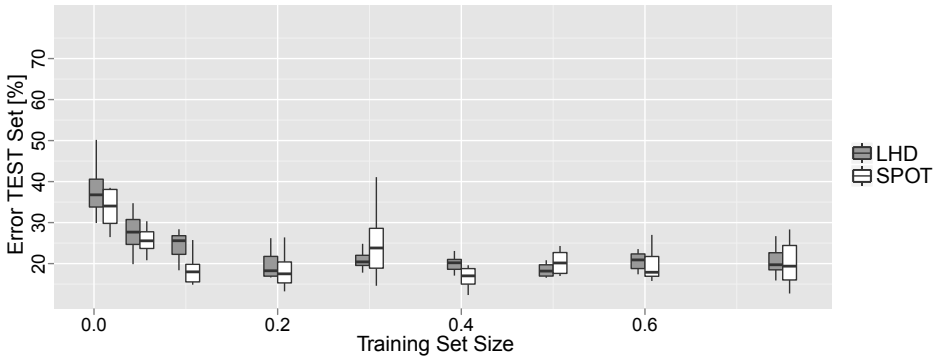

Fig. 2. Comparison of SPOT and LHD algorithms on AppAcid.

obtained when we perform the tuning with the complete training set size (right plots). These plots both exhibit a clear and deep minimum near $\gamma \approx 0$, relatively independent of the cost term. The landscape for $\gamma > 0$ is however very different. With SPOT, we obtain very spiky surrogate models (Fig. 3, upper right). This especially occurs in regions where only few design points are sampled (these are the regions presumably not containing the optimum). We think that when there is a region with a high density of points but large noise in the objective function, GP assumes a very small correlation length leading to spikes in the low-density regions. Overall this leads to a less robust surrogate model. We will show in a forthcoming contribution [9] that slightly different SPOT initial design settings can lead to instable results.

LHD with its equal-density sampling in design space does not have this problem: The landscape (Fig. 3, lower right) exhibits a low curvature and is stable in all experiments. Nevertheless the main issue of LHD sampling, which is the bad scalability for higher dimensions, remains, and this is the reason why this sampling method is less preferable for higher-dimensional search spaces.

We can conclude that if we use a robust surrogate model and a sufficient initial design size of the tuning algorithm, we can obtain good parameter settings very quickly.

4.3 Computation Time

The characteristics of the computation times for different training set sizes are shown in Fig. 4 for the AppAcid dataset. Note that the curve which appears to be linear here can have a different look on other datasets. The roughly linear slope has its reason in the model building process for the AppAcid task. This process includes several other operators beneath SVM training, e.g., preprocessing (principal component analysis and feature selection). In other cases the pure SVM training might grow quadratically with the number of training samples, leading to even larger computation time savings.

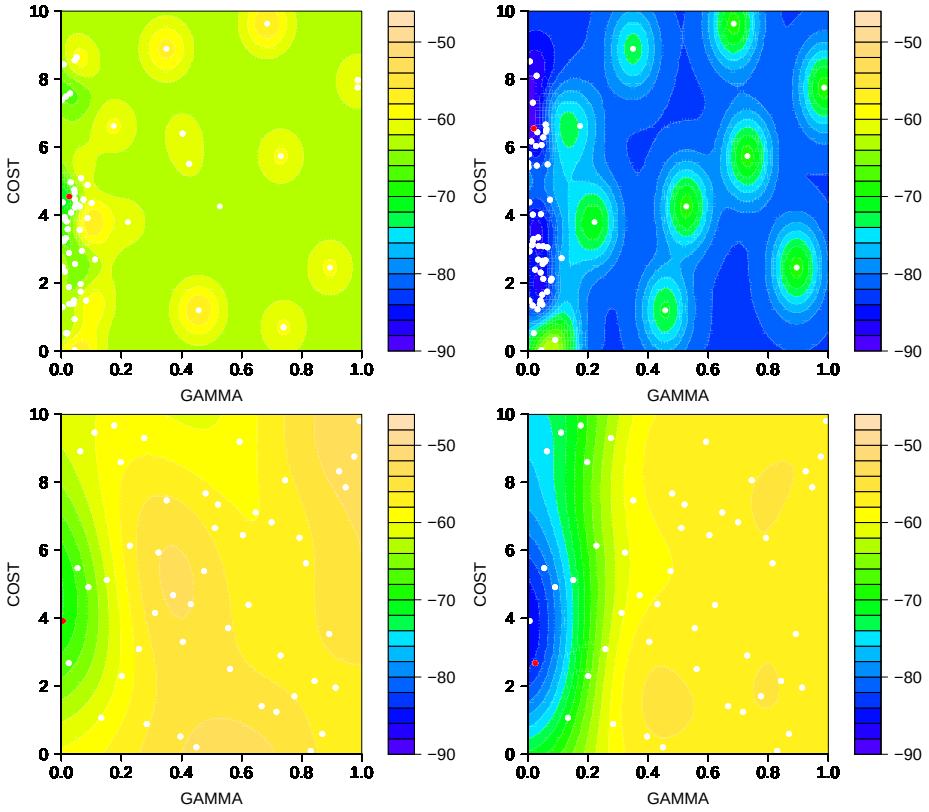

Fig. 3. Contour plots for Sonar. SPOT (top plots) and LHD (bottom plots), using 3 repeats, GP for the contour surface and small (X=10%, left) and large (X=75%, right) training set sizes. White points are design points, the red point is the optimum found by the optimization procedure.

5 Conclusion

We showed that tuning with a low training set size very often leads to good results: An astonishing small fraction of the training set (10-15%) was sufficient to find with high probability a good DM model when performing one 'full' training with the best design point parameters. The resampling strategy (B) (see Sec. 3.3) might be a crucial ingredient for this success with small training samples, but further research is needed to justify this claim.[4]

This study investigated seven different data sets with sizes from 208 to 50000 records. Especially for the bigger datasets large speedups (factor 6 to 10) are possible as Fig. 4 shows. Summarizing the results, we can conclude that both hypotheses **H1** and **H2** hold for the datasets used in this study. In the future we plan to search strategies for selecting the right sample sizes automatically.

[4] We note in passing that Daelemans et al. [4] got negative results with a small training sample, but only on a specific word disambiguation task and without resampling.

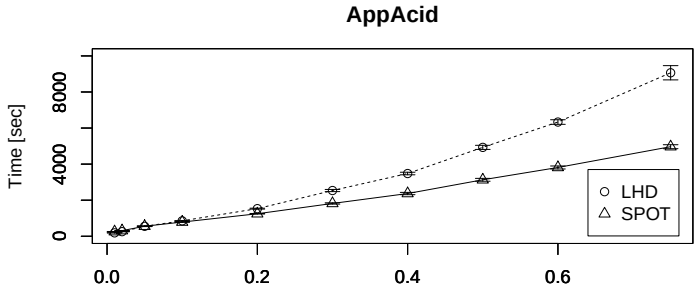

Fig. 4. Computation time on AppAcid as a function of training set size X.

References

1. T. Bartz-Beielstein, C. Lasarczyk, and M. Preuss. The SPO Toolbox. In Bartz-Beielstein et al., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, pages 337–360. Springer, Berlin, Heidelberg, New York, 2010.
2. B. Bischl, O. Mersmann, and H. Trautmann. Resampling methods in model validation. In *Proc. WEMACS 2010, joint to PPSN 2010, Krakow*, page 14, 2010.
3. W. G. Cochran. *Sampling techniques*. Wiley-India, 2007.
4. W. Daelemans, V. Hoste, et al. Combined optimization of feature selection and algorithm parameters in machine learning of language. In *Machine Learning: ECML 2003*, pages 84–95, 2003.
5. B. Efron. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.
6. N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, 2001.
7. A. Jain and D. Zongker. Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on PAMI*, 19(2):153–158, 1997.
8. A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. kernlab — An S4 package for kernel methods in R. Technical report, Department of Statistics and Mathematics, WU Vienna University of Economics and Business, Vienna, 2004.
9. P. Koch and W. Konen. Stability issues in tuning very noisy functions. Submitted to PPSN'2012 Workshop on Automated Selection and Tuning of Algorithms, 2012.
10. W. Konen, P. Koch, O. Flasch, T. Bartz-Beielstein, M. Friese, and B. Naujoks. Tuned data mining: A benchmark study on different tuners. In *Proc. GECCO'2011, Dublin*, pages 1995–2002. ACM, 2011.
11. R. V. Lenth. Some practical guidelines for effective sample size determination. *The American Statistician*, 55(3):187–193, 2001.
12. M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, pages 239–245, 1979.
13. E. Osuna, R. Freund, and F. Girosi. Training support vector machines: an application to face detection. In *IEEE Proc. CVPR'1997*, pages 130–136, 1997.
14. S. J. Raudys and A. K. Jain. Small sample size effects in statistical pattern recognition. *IEEE Transactions on PAMI*, 13(3):252–264, 1991.
15. B. Schölkopf and A. J. Smola. *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. the MIT Press, 2002.
16. V. Vapnik. Statistical learning theory. 1998, 1998.