Ideal kernel tuning: fast and scalable selection of the radial basis kernel spread for support vector classification

Ziad Akram-Ali-Hammouri, Manuel Fernández-Delgado*,
Audi Albtoush, Eva Cernadas, Senén Barro

Centro Singular de Investigación en Tecnoloxías Intelixentes da USC (CiTIUS)
R/Jenaro de la Fuente Domínguez, CP 15782, Santiago de Compostela, Spain

Abstract

A simple and fast method, named ideal kernel tuning, is proposed to select the radial basis kernel spread of the support vector machine for classification. The spread is selected directly from data, without any training nor test, in order to bring the kernel matrix nearer to the ideal kernel matrix, with values one for patterns of the same class and zero otherwise. To avoid scaling with the training set size, the kernel matrix is calculated for a small set of class prototypes. The selected spread can be used also for multi-class classification problems considering the two most populated classes. Compared to other 5 popular algorithms: 1) it achieves state-of-the-art performance; 2) it is 2-4 orders of magnitude faster; and 3) it requires very little memory. Our approach is able to classify large datasets up to 70,000 training patterns.

Keywords: Support vector machine, classification, model selection, RBF Kernel, spread tuning

1. Introduction

The support vector machine (SVM) is a very popular classifier [1] with state-of-the-art performance, particularly with the radial basis function (RBF) kernel [2]. Before the SVM training, two hyper-parameters must be set: regularization (λ) and kernel spread (σ), that measures how large the differences between patterns must be to drive the kernel function towards zero. The optimal σ value depends on the dataset properties and is, in general, different for each dataset. Sub-optimal values often lead to an important loss in the SVM performance. The spread tuning is usually performed by "grid-search", i.e., training the SVM with several σ values from a pre-specified collection and selecting the one with the best performance on a separate test set. The experimental literature has proven that the collection proposed in [3] is adequate for a vast majority of the applications. This highly simplifies the SVM usage because only a repetitive train-test loop must be performed, and no expert knowledge is required to achieve a fine tuning. However, to train and test the SVM many times is very time-consuming for large datasets.

The literature about "SVM model selection" reports alternative approaches for spread selection. The kernel density estimation (KDE) selects the σ value that maximizes a function of disimilarity between two classes that is based on the RBF kernel [4]. Genetic algorithms are used in [5] to determine of λ and σ by optimizing the test performance over small datasets (up to 768 training patterns). The paper [6] also uses genetic optimization to find the hyper-parameter values that maximize the relative

*Corresponding author
Email address: manuel.fernandez.delgado@usc.es
(Manuel Fernández-Delgado)

efficiencies of binary SVMs in multi-class problems up to 4,400 training patterns. Evolutionary algorithms are used in [7] to perform surrogate-assisted multi-objective minimization of bias and variance, estimated by training and testing the SVM. The Bat algorithm [8] is a bio-inspired method based on genetic optimization that iteratively searches for λ and σ selecting the values that minimize the classification error of the SVM. Despite of exhibit early convergence compared to other meta-heuristic methods, it requires to train and test the SVM inside the optimization loop, that reduces its computational efficiency and hinders its scalability for large datasets (only up to 1,000 training patterns and 60 inputs). The same authors [9] combine the social ski driver algorithm and the synthetic minority over-sampling technique (SMOTE) to select σ for imbalanced datasets. A dynamic strategy based on particle swarm optimization is proposed in [10] for the simultaneous selection of λ and σ and applied with datasets up to 7,000 training patterns and 256 inputs. Segmented dichotomy is combined with grid search in [11] to select both hyper-parameters, but it requires to train and test the SVM iteratively, so it was applied to medium size datasets up to 3,000 training patterns. Active testing was used in [12] to select both hyper-parameters from dataset properties such as training set size, number of inputs and classes, mean correlation among inputs, input skewness and kurtosis, class entropy, requiring the SVM execution for each different candidate solution. A method based on the Fisher criterion is proposed in [13] to select both hyper-parameters in order to maximize the between-class similarity and minimize the within-class separability, estimating this separability by cosine similarity in the kernel space.

Similarly to grid-search, most of the previous approaches require to train and test the SVM several times, often under the scope of some optimization method, that does not scale with the training set size. The goal of the current paper is to develop a method able to find the optimal σ directly from the data (patterns and class labels) without the need to train and test the SVM, that does not scale to large datasets. Our proposal, named ideal kernel tuning (IKT), is to select the σ value that brings the RBF kernel the closest possible to the ideal kernel [14]. The basic idea of this method was already used in the fast support vector classifier [14], but the current paper performs exhaustive numerical experiments on benchmark datasets that compare IKT to other state-of-the-art approaches for model selection. The paper is organized as follows: section 2 presents the proposed method, whose experimental results are reported and discussed in section 3, and section 4 summarizes the conclusions of this study.

2. Ideal kernel tuning

The SVM kernel $K(\mathbf{x}, \mathbf{y})$ is a generalized similarity measure between two patterns $\mathbf{x}$ and $\mathbf{y}$. This similarity can be considered as their dot product after being projected into a high-dimensional feature space by a mapping $\mathbf{\Phi}(\mathbf{x})$, so that $K(\mathbf{x}, \mathbf{y}) = \mathbf{\Phi}(\mathbf{x})^T \mathbf{\Phi}(\mathbf{y})$. The RBF kernel is a Gaussian function of spread σ applied over the difference between both patterns:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{|\mathbf{x} - \mathbf{y}|^2}{2\sigma^2}\right) \tag{1}$$

where the spread σ must be selected before the SVM training stage. Let us consider a binary classification problem with C=2 classes, N training patterns $\mathbf{x}_n$ of dimensionality I and class label $c_n \in \{1, \ldots, C\}$, with $n = 1, \ldots, N$. The RBF kernel matrix $\mathbf{K}$ is a N-order square matrix with elements $K_{nm} = K(\mathbf{x}_n, \mathbf{x}_m)$, with $n, m = 1, \ldots, N$, being K defined by eq. 1. On the other hand, the ideal kernel [15] matrix $\mathbf{I}$, also squared of order N, has elements:

$$I_{nm} = I(\mathbf{x}_n, \mathbf{x}_m) = \begin{cases} 1 & c_n = c_m \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The ideal kernel defines a class-based similarity that discriminates perfectly both classes, but its analytical expression depends on the dataset. This concept is useful because defines how a kernel should be in order to achieve perfect classification. Selecting the σ that provides the RBF kernel nearest to the ideal kernel should provide the best classification performance. This is achieved minimizing the mean absolute difference between matrices $\mathbf{K}$ and $\mathbf{I}$. The **ideal kernel tuning** (IKT), that is detailed in algorithm 1, is based on this idea.

Algorithm 1: Ideal kernel tuning.

Algorithm: IKT$(\{\mathbf{x}_n, c_n\}_{n=1}^N)$

Data: $\{\mathbf{x}_n, c_n\}_{n=1}^N$, $\mathbf{x}_n$; $c_n \in \{1, \ldots, C\}$
Result: σ^*: optimal RBF kernel spread

$\{N_i \leftarrow 0\}_{i=1}^C$; $L \leftarrow 100$
$\{\mathbf{p}_{il} \leftarrow \mathbf{0}; L_i, N_{il} \leftarrow 0\}_{il=1}^{C,L}$
for $n \leftarrow 1, N$ **do**
  $i \leftarrow c_n$; $N_i \leftarrow N_i + 1$
  if $L_i < L$ **then**
    $l \leftarrow L_i$; $\mathbf{p}_{il} \leftarrow \mathbf{x}_n$; $L_i \leftarrow L_i + 1$
  else
    $r \leftarrow \arg\min_{l=1\ldots,L} |\mathbf{p}_{il} - \mathbf{x}_n|$; $N_{ir} \leftarrow N_{ir} + 1$
    $\mathbf{p}_{ir} \leftarrow \left(1 - \frac{1}{N_{ir}}\right)\mathbf{p}_{ir} + \frac{\mathbf{x}_n}{N_{ir}}$
  end
end
$a \leftarrow \arg\max_{i=1,\ldots,C}\{N_i\}$; $b \leftarrow \arg\max_{i=1..C, i\neq a}\{N_i\}$
$U \leftarrow L_a + L_b$; $\{I_{ls} \leftarrow 0\}_{ls=1}^U$
$\{I_{ls} \leftarrow 1\}_{ls=1}^{L_a}$; $\{I_{ls} \leftarrow 1\}_{ls=L_a+1}^U$
$\{\mathbf{z}_l \leftarrow \mathbf{p}_{al}\}_{l=1}^{L_a}$; $\{\mathbf{z}_l \leftarrow \mathbf{p}_{b(l-L_a)}\}_{l=L_a+1}^U$
$\left\{K_{ls}(\sigma) \leftarrow \exp\left(\frac{-|\mathbf{z}_l - \mathbf{z}_s|^2}{2\sigma^2}\right)\right\}_{ls=1}^U$
$R \leftarrow 19$; $\alpha \leftarrow 2^{15/2}$; $\{\sigma_i \leftarrow \alpha 2^{-i/2}\}_{i=1}^R$
$\left\{D_i \leftarrow \frac{1}{U^2}\sum_{l=1}^U\sum_{s=1}^U |K_{ls}(\sigma_i) - I_{ls}|\right\}_{i=1}^R$
$j \leftarrow \arg\min_{i=1,\ldots,R}\{D_i\}$; $\delta \leftarrow |D_R - D_1|/10$
if $|D_j - D_1| < \delta$ `or` $|D_R - D_j| < \delta$ **then**
  for $i \leftarrow 1, R$ **do**
    $q \leftarrow \max(1, i-2)$; $r \leftarrow \max(i+2, R)$
    $F_i \leftarrow$ `median`$(\{D_k\}_{k=q}^r)$
  end
  $\{G_i \leftarrow F_{i+1} - F_i\}_{i=1}^{R-1}$; $j \leftarrow \arg\max_{i=1,\ldots,R-1}\{G_i\}$
end
$\sigma^* \leftarrow \sigma_j$

In principle, this procedure is designed for binary classification. However, the multi-class SVM uses several binary SVMs with equal σ. This value can be selected for only the two most populated classes (denoted as a and b), and used for all the binary SVMs. Considering only the N_a and N_b training patterns of classes a and b, respectively, both the RBF and ideal kernel matrices $\mathbf{K}$ and $\mathbf{I}$ are of size M^2, with $M = N_a + N_b$. The calculation of this matrix becomes slow and drives out of memory in large datasets with high N_a and N_b. In order to avoid this drawback, the M training patterns of classes a and b are replaced by a reduced set of class prototypes $\{\mathbf{p}_{il}\}_{l=1}^{L_i}$, with $L_i = \min(N_i, L) \leq L$, where N_i is the number of patterns of class $i = a, b$, while L is the upper bound to the number of prototypes per class, set to a low value. It must be $U = L_a + L_b \leq 2L \ll M$, i.e., the number of prototypes should be much less than the number M of training patterns of classes a and b. Thus, RBF and ideal kernel matrices $\mathbf{K}$ and $\mathbf{I}$ are of size U^2.

To calculate the class prototypes, the first L training patterns $\mathbf{x}_n$ of class i are stored as starting prototypes $\{\mathbf{p}_{il}\}_{l=1}^L$. When there are $N_i < L$ patterns of class i, only N_i prototypes are created, so that $L_i = N_i$ and no prototype is updated. When $N_i \geq L$, after creating the $L_i = L$ starting prototypes each pattern $\mathbf{x}_n$ of class $i = c_n$ updates its nearest prototype $\mathbf{p}_{ir}$ of the class i to which $\mathbf{x}_n$ belongs, according to:

$$\mathbf{p}'_{ir} = \left(1 - \frac{1}{N_{ir}}\right)\mathbf{p}_{ir} + \frac{\mathbf{x}_n}{N_{ir}}, \quad i = c_n \qquad (3)$$
$$r = \arg\min_{l=1\ldots,L_i} |\mathbf{p}_{il} - \mathbf{x}_n|, \quad N'_{ir} = N_{ir} + 1$$

where $\mathbf{p}_{ir}$ and $\mathbf{p}'_{ir}$ are the old and new versions, respectively, of the r-th prototype of class i, which is the nearest prototype to $\mathbf{x}_n$, while N_{ir} and N'_{ir} are the old and new numbers of training patterns used to update $\mathbf{p}_{ir}$. Although the first L patterns of a class were similar among them, new patterns would update their nearest prototypes and finally the prototype collection $\{\mathbf{p}_{il}\}_{l=1}^L$ of class i will represent its training patterns.

Although the differences $|\mathbf{p}_{il} - \mathbf{x}_n|$ must be calculated for each training pattern and the L_i ($\leq L$) prototypes of class $i = c_n$, the number L is upper bounded, so the number of calculations keeps low for large datasets. Besides, matrices $\mathbf{K}$ and $\mathbf{I}$ are small because their size U^2 verifies $U^2 \leq (2L)^2$. After some previous experiments, we saw that $L = 100$ prototypes is enough to represent a class in large datasets where $L_a = L_b = L$ and $U = 2L$, keeping the kernel matrix size below $(2L)^2 = 40,000$. However, when the number of inputs I or classes C is large, a lower L should be set to avoid slowness and excesive memory

Table 1: Number of patterns (T), training patterns (N), inputs (I) and classes (C) of the classification problems.

No.	Dataset	Patterns T	Train N	Inputs I	Classes C
1	tissue	106	58	9	6
2	hepatitis	155	78	24	2
3	wine	178	90	13	3
4	sonar	208	106	60	2
5	seeds	210	108	7	3
6	heart	270	136	20	2
7	voting	342	218	32	2
8	ionosphere	350	178	33	2
9	dermatology	366	186	98	6
10	german	366	500	44	2
11	monks	432	216	17	2
12	wdbc	569	286	30	2
13	synthetic	600	300	60	6
14	australian	690	346	35	2
15	energy	768	386	8	3
16	pima	768	384	8	2
17	vehicle	846	426	18	4
18	annealing	886	450	52	5
19	tic	958	480	18	2
20	mammograph	961	482	5	2
21	isolet	1,559	780	617	26
22	imseg	2,086	140	18	7
23	abalone	4,177	2,090	9	3
24	sat	6,435	3,222	36	6
25	musk	6,598	3,302	166	2
26	usps	9,298	4,650	256	2
27	grid	10,000	5,000	13	2
28	nursery	12,958	6,480	27	4
29	magic	19,020	9,510	10	2
30	letter	20,000	10,018	16	26
31	chess	28,056	14,044	40	18
32	adult	48,842	24,422	105	2
33	shuttle	57,977	43,483	9	5
34	mnist	70,000	60,000	784	10
35	wisdm	73,803	55,380	92	18
36	ijcnn1	141,691	70,848	22	2
37	poker	1,025,010	25,010	10	2

requirements. The elements $\{K_{ls}\}_{ls=1}^{U}$ of matrix $\mathbf{K}$ are:

$$K_{ls}(\sigma) = K(\mathbf{z}_l, \mathbf{z}_s) = \exp\left(-\frac{|\mathbf{z}_l - \mathbf{z}_s|^2}{2\sigma^2}\right) \quad (4)$$

where the vectors $\{\mathbf{z}_l\}_{l=1}^{U}$ are defined by:

$$\mathbf{z}_l = \begin{cases} \mathbf{p}_{al} & l = 1, \ldots, L_a \\ \mathbf{p}_{b(l-L_a)} & l = L_a + 1, \ldots, U \end{cases} \quad (5)$$

where $\{\mathbf{p}_{al}\}_{l=1}^{L_a}$ and $\{\mathbf{p}_{bl}\}_{l=1}^{L_b}$ are the prototypes of classes a and b. The ideal kernel matrix $\mathbf{I}$, also of size U^2, has elements defined by $I_{ls} = 1$ for $l, s \in \{1, \ldots, L_a\}$ or $l, s \in \{L_a + 1, \ldots, U\}$, i.e. both prototypes l and s are of the same class, and $I_{ls} = 0$ otherwise, i.e. both prototypes are of different classes. The IKT selects σ to minimize the mean absolute difference between $\mathbf{K}$ and $\mathbf{I}$. In order to avoid a computationally intensive optimization, the search is limited to the $R = 19$ values defined by the Libsvm library [3] as $\sigma_i = \alpha 2^{-i/2}$, with $\alpha = 2^{15/2}$ and $i = 1, \ldots, R$. The matrix difference, denoted as D_i, for σ_i is calculated as:

$$D_i = \frac{1}{U^2} \sum_{l=1}^{U} \sum_{s=1}^{U} |K_{ls}(\sigma_i) - I_{ls}| \quad (6)$$

and the index j that minimizes $\{D_i\}_{i=1}^{R}$ is selected as:

$$j = \arg\min_{i=1,\ldots,R} \{D_i\} \quad (7)$$

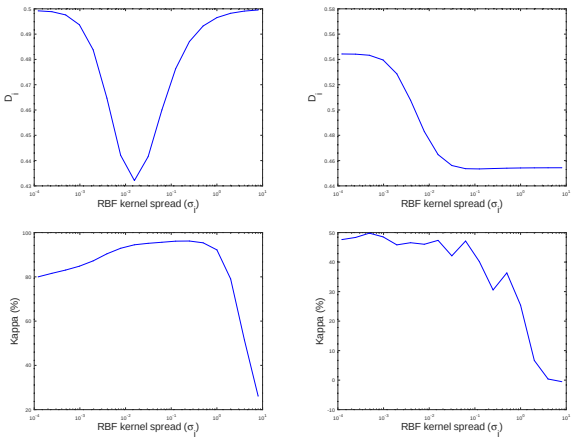

Figure 1: Difference D_i (upper panel) and performance (lower panel) vs. σ_i for datasets `letter` and `pima`.

Figure 1 plots $\{D_i\}_{i=1}^{R}$ (upper panel) and classification performance (measured by the Cohen kappa statistic, lower panel) for two datasets (`letter` and `pima`) included in the experimental work (section 3). The left profile is the most common, with a minimum of D_i (upper left) in the values of σ_i with the highest performance (lower left). However, in some datasets D_i looks as the upper right plot, and the highest performance (lower right) is achieved in the σ_i with the highest slope of D_i (upper right). In this case, the minimum D_i is very near to D_1 or D_R, that

happens when $|D_j - D_1| < \delta$ (minimum at D_1) or $|D_R - D_j| < \delta$ (minimum at D_R), with j calculated in eq. 7, being $\delta = |D_R - D_1|/10$ a threshold (line 22 of algorithm 1). In this case, a median filter of radius 2 is applied to $\{D_i\}_{i=1}^{R}$ for noise reduction. The smoothed filter output is $F_i = \texttt{median}(\{D_k\}_{k=q}^{r})$, with $q = \max(1, i-2)$, while $r = \min(i+2, R)$ and $i = 1, \ldots, R$. Since the best σ is on the highest slope, the derivative $G_i = F_{i+1} - F_i$ of F_i is calculated, with $i = 1, \ldots, R-1$, and j is set to the index i that maximizes G_i:

$$j = \underset{i=1,\ldots,R-1}{\arg\max} \{G_i\} \tag{8}$$

Finally, the selected spread value is $\sigma^* = \sigma_j$, with j defined by eq. 7 or 8.

3. Results and discussion

The IKT was programmed in the Octave[1] scientific computing language version 5.2. The σ selected by IKT is used to train and test the SVM implemented by the Libsvm library [16] with λ=100 using 4-fold cross validation with 3 folds for training (excepting grid-search, see below) and 1 fold for test in each trial. We compared IKT with several state-of-the-art methods for selecting the RBF kernel spread: 1) kernel density estimation, **KDE**, described in [4], applied on the two most populated classes; 2) the classical grid-search procedure, **GS**, that selects the σ with the highest performance over a validation set with the 50% of the original training set; 3) a genetic algorithm, **GA**, implemented in the R statistical computing language[2] by the `GA` package; 4) particle-swarm optimization, **PSO**, implemented by the `pso` package; and 5) Bayesian optimizer, **Bayes**, implemented by the `rBayesianOptimization` package [8]. The KDE and GS are programmed in Octave, while GA, PSO and Bayes optimize the classification performance of the SVM implemented by the `kernlab` package. The six methods were executed on a collection of 37 datasets (Table 1) selected from the UCI Machine Learning Repository [17] with up to 70,000 training patterns, 784 inputs and 26 classes. The experiments ran on a desktop computer with the following features: Intel Core i7-9700K processor (8 cores) at 3.6GHz with 64GB RAM under Kubuntu 20.04. The performance, measured by the Cohen kappa statistic [18], the time elapsed by the selection of the kernel spread and the memory required by the selection method were recorded for each method.

Table 2: Kappa (in %).

Dataset	IKT	KDE	GS	GA	PSO	Bayes
tissue	58.0	64.9	61.1	59.6	61.7	64.8
hepatitis	39.3	35.4	42.4	30.4	30.4	31.5
wine	97.5	95.0	97.5	96.6	96.6	95.8
sonar	76.9	71.8	68.9	77.9	58.5	76.0
seeds	91.3	90.5	90.6	88.5	89.8	89.2
heart	53.5	53.5	55.7	61.8	67.0	66.2
voting	89.8	89.8	88.9	87.3	86.0	89.4
ionosphere	84.1	83.5	81.7	84.7	87.1	83.5
dermatology	95.6	95.6	96.6	95.9	96.2	95.9
german	28.1	29.1	28.5	34.0	38.3	41.3
monks	100.0	100.0	100.0	100.0	100.0	100.0
wdbc	89.1	88.4	86.9	85.1	84.8	88.6
synthetic	99.2	99.4	98.8	99.0	99.0	98.4
australian	63.1	63.1	57.8	64.4	67.9	67.2
energy	90.4	90.4	92.3	91.2	92.9	92.5
pima	30.2	40.4	28.0	38.0	47.3	45.5
vehicle	78.1	78.1	76.5	79.4	75.4	75.6
annealing	98.6	98.3	98.1	97.8	97.2	97.5
tic	97.2	97.7	97.2	97.5	97.0	96.8
mammograph	62.1	64.1	46.4	66.3	63.5	63.9
isolet	95.0	95.1	94.5	79.4	93.6	93.6
imseg	90.7	89.9	91.0	89.3	83.9	89.6
abalone	32.9	32.6	23.6	32.6	32.6	33.0
sat	89.4	89.2	89.0	89.5	89.5	89.1
musk	98.9	99.2	98.7	99.0	99.0	99.1
usps	97.4	97.3	96.8	90.6	95.7	97.3
grid	96.8	97.9	97.6	98.1	98.0	98.7
nursery	100.0	100.0	100.0	100.0	100.0	100.0
magic	71.0	71.0	63.3	71.0	70.9	70.2
letter	96.8	96.8	97.0	97.4	97.4	97.4
chess	89.0	88.6	88.8	89.3	89.4	89.3
adult	52.9	53.6	18.0	–	56.0	56.0
shuttle	99.7	99.7	99.8	–	99.8	99.8
mnist	83.5	83.5	80.7	–	–	80.7
wisdm	89.1	75.0	90.3	–	–	90.3
ijcnn1	95.4	–	81.8	–	–	95.7
poker	13.4	17.6	21.8	–	22.9	21.8
Average	78.8	78.2	76.4	79.7	78.4	80.0

Table 2 reports the kappa achieved by the 6 methods. The missing values correspond to datasets where the method failed due to out-of-memory errors or did not finish within 6 days. The average values of the last line are calculated, for each method, over all the

[1] http://www.octave.org (July,2021)
[2] http://www.r-project.org (July, 2021)

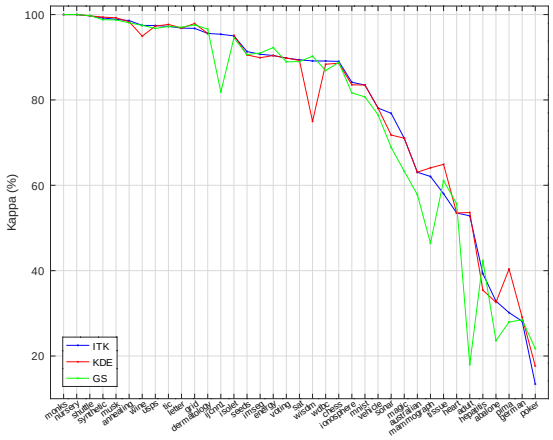

Figure 2: Kappa (in %) achieved by IKT, KDE and GS.

Table 3: Elapsed time (in s.).

Dataset	IKT	KDE	GS	GA	PSO	Bayes
tissue	0.005	0.010	0.023	2.0	0.9	535
hepatitis	0.010	0.035	0.048	3.2	1.2	319
wine	0.009	0.028	0.038	2.5	1.0	220
sonar	0.032	0.060	0.079	7.3	2.5	152
seeds	0.008	0.039	0.025	2.0	0.9	196
heart	0.021	0.086	0.063	6.7	2.2	128
voting	0.031	0.212	0.075	9.1	2.7	137
ionosphere	0.035	0.143	0.080	7.9	3.1	230
dermatology	0.052	0.085	0.324	29.8	8.8	23.3
german	0.059	0.540	0.573	48.5	11.6	19.8
monks	0.048	0.245	0.350	8.7	4.5	140
wdbc	0.055	0.211	0.157	17.0	3.5	18.9
synthetic	0.067	0.080	0.324	37.3	10.5	65.1
australian	0.091	0.313	0.309	27.7	7.4	20.8
energy	0.042	0.260	0.178	24.4	4.6	88.9
pima	0.041	0.367	0.276	19.3	3.8	30.8
vehicle	0.074	0.128	0.396	37.5	7.8	41.9
annealing	0.064	0.427	0.786	53.5	12.4	180
tic	0.046	0.486	0.925	51.4	18.4	132
mammograph	0.030	0.429	0.517	27.3	6.7	68.9
isolet	1.47	0.720	42.4	2793	1109	313
imseg	0.014	0.020	0.061	3.9	1.7	8.6
abalone	0.100	3.19	8.66	574	161	78.5
sat	0.442	3.69	15.9	2149	439	101
musk	2.06	18.0	123	8872	1244	279
usps	4.50	41.5	1336	36154	3143	1111
grid	0.290	36.6	46.5	1423	225	59.4
nursery	0.459	25.1	140	7257	2226	332
magic	0.470	137	161	13673	2503	218
letter	0.819	1.40	126	26796	4378	361
chess	1.28	28.4	753	70106	14253	1208
adult	10.4	1967	17215	–	85099	17155
shuttle	1.43	2921	18.6	–	469	101
mnist	105	1299	85583	–	–	163685
wisdm	11.4	30.1	4436	–	–	18067
ijcnn1	384	–	12493	–	–	16029
poker	0.775	670	733	–	12686	1017
Average	–	105	163	5549	1760	6420

datasets excluding those with fails. The IKT, GS and Bayes never fail, while KDE fails on dataset `ijcnn1`, PSO in three datasets (`mnist`, `wisdm` and `ijcnn1`) and GA in the 6 largest datasets. The values for each datasets are in general very similar for the different methods, with some exceptions: GS achieves 18% in dataset `adult`, where IKT, KDE, PSO and Bayes achieve 52-56%; and GA achieves 79.4% in `isolet`, while the others achieve 94%. The average values are also similar, being Bayes the best (80%), followed by GA (79.7%), IKT (78.8%), PSO (78.4%) and KDE (78.2%), while GS performs slightly worse (76.4%). Figure 2 plots the kappa values of IKT, KDE and GS, sorted by decreasing values. The GS is below the other methods in four datasets (`ijcnn1`, `mammograph`, `adult` and `abalone`), while KDE is twice above (datasets `tissue` and `pima`) and once below IKT (dataset `wisdm`).

Table 3 reports the time spent by each method to select σ, excluding the SVM training and test. The last row reports, for all the methods excepting IKT and for each dataset, the ratio of the time spent by that method on that dataset divided by the time spent by IKT on the same dataset, averaged over all the datasets excluding fails. This average says how many times the method is slower than IKT. The times spent by IKT are very small and raise very slowly with the dataset size. In some cases the times are higher compared to datasets with similar size: `isolet` (1.47 s. while the previous dataset `mammograph` spends 0.03 and the next dataset `imseg` spends 0.014), because the high I and C slow down the prototype learning; `musk`, `usps` and `adult`, also due to their high I; `mnist` due to the large $I = 784$ and $C = 10$. The KDE is also fast, but slower than IKT, in some cases with high difference: in `usps`, `grid` and `nursery`, KDE spends 41, 36 and 25 s., while IKT spends 4.5, 0.29 and 0.45 s. Larger differences occur in `adult`, where IKT and KDE spend 10.4 and 1,967 s., and `shuttle` (1.43 and 2,921 s.), because the two most populated classes (1 and 4)

have 40,856 of 43,483 training patterns, so KDE is much more slower than IKT, that uses a reduced number of class prototypes. In fact, KDE fails in dataset `ijcnn1` by lack of memory. Given that GS requires to train and test the SVM many times, it is much slower than IKT and KDE, specially in datasets `adult` (17,215 s. with GS, 10.4 and 1,967 s. with IKT and KDE) and `mnist` (85,583 s. with GS, 105 and 1,299 s. with IKT and KDE). However, GS is much faster than KDE in dataset `shuttle` (18.6 and 2,921 s., respectively), because the SVM training in the former is relatively fast, while the latter is slow as we explained above. The GA is clearly the slowest one, in fact it does not finish after days of execution in the six largest datasets. The PSO is also slow, but faster than GA in all the datasets, failing only in the three largest datasets. Finally, the Bayes method is slower than GA and PSO in the small datasets but faster in the largest ones, and it does not fail in any dataset, as opposite to BA and PSO. In average over all datasets, KDE and GS are 105 and 163 times (two orders of magnitude) slower than IKT, while GA, PSO and Bayes are 5,549, 1,760 and 6,420 times (three-four orders of magnitude) slower than IKT. The number corresponding to Bayes is somehow confusing, because it seems to be the slowest one, while GA and PSO are much slower than Bayes for the largest datasets. Similarly to GS, the large times of GA, PSO and Bayes is expectable because they require to train and test the SVM.

Figure 3 plots the time spent by IKT, KDE and GS in all the datasets. The plots of KDE and GS are almost always above IKT, with the only exception of `isolet`, where the high C slows down the prototype learning of IKT (1.47 s.), that is slower than KDE (0.72 s.). Generally, IKT is between one and two orders of magnitude faster than KDE and GS. To avoid confusion, the times of GA, PSO and Bayes are not plotted, but they are much higher in all the datasets.

Table 4 reports the RAM memory required by each method and dataset, excluding the memory used by the SVM training and prediction, that is much larger. Thanks to the limitation on the number of class prototypes, the memory required by IKT is below 10 MB even in the largest datasets, much below the capability of a today's standard desktop computer. However, the memory requirements of KDE raise in dataset `abalone` (N=2,090, memory of 70.9 MB) reaching large values in datasets `adult` (20.5 GB), `shuttle` (25.4 GB), `mnist` (2.6 GB) and `poker` (9.5 GB), due to the high number of training patterns of the two most populated classes: 24,422 in `adult`, 40,856 in `shuttle`, 12,873 in `mnist` and 25,010 in `poker`. The GS uses also more memory than IKT, specially in datasets larger than `musk`. The GA and PSO require small amounts of memory. Bayes requires less memory than IKT in the small datasets, and more memory in the large datasets: `adult` (29.5 MB and 8.2 MB with Bayes and IKT), `mnist` (344 MB and 9.7 MB), `wisdm` (39.1 MB and 3.6) and `ijcnn1` (18.3 MB and 4.5 MB). The last line reports the ratio of the memory requirements of each method divided by IKT excluding datasets with fails: the KDE, GS and Bayes require 400, 4 and 2 times more memory, respectively, than IKT. The memory of GA and PSO is so low as IKT on the small datasets, being unknown on the largest datasets because they fail. Figure 4 plots the memory consumption of the six methods. In the small datasets GA, PSO and Bayes are below IKT, but in the large datasets GS and PSO do not run and Bayes overcomes IKT, while KDE is the most expensive and GS also overcomes IKT.

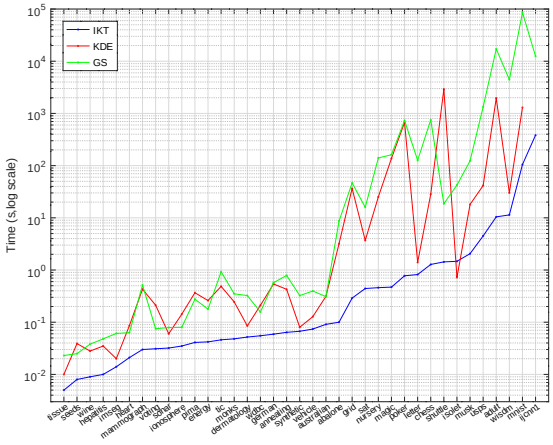

Figure 3: Time (in s.) spent by IKT, KDE and GS.

Table 4: Memory (in MB).

Dataset	IKT	KDE	GS	GA	PSO	Bayes
tissue	0.04	0.02	0.02	0.04	0.04	0.04
hepatitis	0.38	0.23	0.05	0.05	0.05	0.05
wine	0.26	0.16	0.05	0.05	0.05	0.05
sonar	0.81	0.45	0.24	0.12	0.12	0.12
seeds	0.28	0.18	0.03	0.04	0.04	0.05
heart	0.98	0.67	0.10	0.06	0.06	0.07
voting	1.0	1.7	0.09	0.06	0.05	0.06
ionosphere	1.1	1.1	0.21	0.11	0.10	0.11
dermatology	1.1	0.43	0.52	0.20	0.20	0.20
german	1.2	8.7	0.41	0.15	0.15	0.16
monks	1.0	1.6	0.10	0.06	0.06	0.06
wdbc	1.1	2.9	0.34	0.14	0.14	0.14
synthetic	1.0	0.42	0.71	0.25	0.25	0.26
australian	1.3	4.3	0.35	0.14	0.14	0.14
energy	1.0	3.5	0.14	0.07	0.07	0.07
pima	0.99	5.1	0.13	0.07	0.07	0.07
vehicle	1.1	1.7	0.32	0.13	0.12	0.13
annealing	1.2	5.5	0.53	0.20	0.20	0.20
tic	1.2	8.0	0.32	0.11	0.11	0.12
mammograph	0.98	8.0	0.09	0.06	0.06	0.07
isolet	11.8	0.55	18.6	5.7	5.7	5.7
imseg	0.16	0.06	0.11	0.07	0.06	0.07
abalone	1.2	70.9	0.70	0.24	0.24	0.24
sat	2.6	80.0	4.7	1.4	1.4	1.4
musk	7.9	380	19.9	6.4	6.3	6.4
usps	8.0	756	45.6	13.7	13.7	13.7
grid	1.9	859	2.7	0.81	0.81	0.81
nursery	3.2	634	4.2	1.1	1.1	1.1
magic	2.3	3106	3.9	1.2	1.2	1.2
letter	3.3	22.6	8.9	1.9	1.9	1.9
chess	8.1	659	13.7	3.3	3.3	3.3
adult	8.2	20505	94.3	–	29.5	29.5
shuttle	4.4	25474	3.7	–	3.4	3.4
mnist	9.7	2660	690	–	–	344
wisdm	3.6	641	80.7	–	–	39.1
ijcnn1	4.5	–	49.6	–	–	18.3
poker	3.5	9547	4.3	–	2.1	2.1
Average	—	383.9	3.9	0.313	0.4	1.8

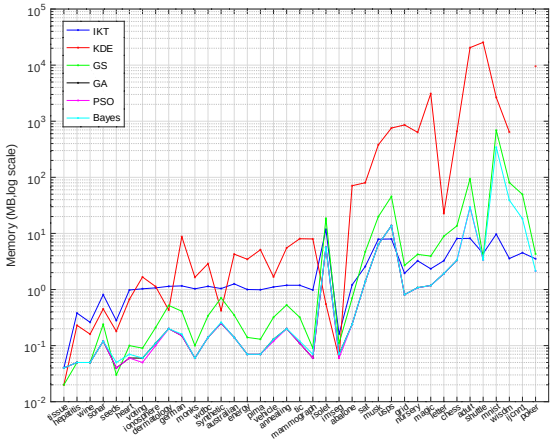

Figure 4: Memory (in MB).

4. Conclusion

We propose the ideal kernel tuning (IKT), a simple and efficient method to select the RBF kernel spread of the SVM for classification. The method selects spread directly from data and is scalable for large datasets because no SVM training nor testing is required. First, the ideal kernel matrix, with value 1 only in the items of the same class and 0 otherwise, is created for the binary classification problem defined by the two most populated classes. Second, the spread that mimimizes the difference between the RBF and ideal kernel matrices is found. Third, both matrices are calculated for a limited number of prototypes of both classes instead of the original training patterns. Since their size is bound, IKT scales well with the training set size, so the time and memory consumption do not raise in large datasets. The spread value can be used in binary or multi-class classification, where it is common practice to be shared among binary SVMs. Our experiments compare IKT to kernel density estimation (KDE), grid-search (GS), genetic algorithms (GA), particle swarm optimization (PSO) and Bayesian optimization on a collection of 37 datasets up to 70,000 training patterns, 784 inputs and 26 classes. The performance of IKT outperforms GS, that is the standard tuning method, KDE and PSO, being only slightly below GA and PSO, that spend many days in the largest datasets. The IKT is between two and four orders faster than the other methods and requires little memory for the capability of today's computers (400 times less than KDE), being interesting for SVM tuning with medium and large datasets. Future work includes to extend this method for regression.

Acknowledgments

This work has received financial support from the Xunta de Galicia (accreditation 2019-2022 ED431G-2019/04) and the European Regional Development Fund (ERDF).

References

[1] C. Cortes, V. Vapnik, Support-vector networks, Mach Learn 20 (3) (1995) 273–297.

[2] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, J Mach Learn Res 15 (1) (2014) 3133–3181.

[3] C.-W. Hsu, C.-C. Chang, C.-J. Lin, et al., A practical guide to support vector classification (2003).

[4] M. Menezes, L. Torres, A. Braga, Width optimization of RBF kernels for binary classification of support vector machines: A density estimation-based approach, Pattern Recogn Lett 128 (2019) 1–7.

[5] F. Friedrichs, C. Igel, Evolutionary tuning of multiple SVM parameters, Neurocomputing 64 (2005) 107–117.

[6] V. Saradhi, K. Girish, Effective parameter tuning of SVMs using radius/margin bound through data envelopment analysis, Neural Process Lett 41 (2015) 125–138.

[7] A. Rosales-Pere, J. Gonzalez, C. Coello, H. Escalante, C. Reyes-Garcia, Surrogate-assisted multi-objective model selection for support vector machines, Neurocomputing 150 (2015) 163–172.

[8] A. Tharwat, A. E. Hassanien, B. E. Elnaghi, A BA-based algorithm for parameter optimization of support vector machine, Pattern Recogn Lett 93 (2017) 13–22.

[9] A. Tharwat, T. Gabel, Parameters optimization of support vector machines for imbalanced data using social ski driver algorithm, Neural Comput Appl 32 (2020) 6925–6938.

[10] M. Kapp, R. Sabourin, P. Maupin, A dynamic model selection strategy for support vector machine classifiers, Appl Soft Comput 12 (8) (2012) 2550–2565.

[11] H. Shi, H. Xiao, J. Zhou, N. Li, H. Zhou, Radial basis function kernel parameter optimization algorithm in support vector machine based on segmented dichotomy, in: Intl Conf Syst Inform (ICSAI), 2018, pp. 383–388.

[12] P. Miranda, R. Prudencio, Active testing for SVM parameter selection, in: Intl J Conf Neural Netw, 2013, pp. 1–8.

[13] Z. Liu, H. Xu, Kernel parameter selection for support vector machine classification, J Alg Comput Technol 8 (2) (2014) 163–177.

[14] Z. Akram-Ali-Hammouri, M. Fernández-Delgado, E. Cernadas, S. Barro, Fast support vector classification for large-scale problems, IEEE T Pattern Anal 43 (9) (2021) 1–1. doi:0.1109/TPAMI.2021.3085969.

[15] M. Pérez-Ortiz, P. Gutiérrez, J. Sánchez-Monedero, C. Hervás-Martínez, A study on multi-scale kernel optimisation via centered kernel-target alignment, Neural Process Lett 44 (2) (2016) 491–517.

[16] C. Chang, C. Lin, LIBSVM: A library for support vector machines, ACM Trans Intel Syst Technol 2 (3) (2011) 1–27.

[17] D. Dua, C. Graff, UCI machine learning repository (2017). URL http://archive.ics.uci.edu/ml

[18] M. McHugh, Interrater reliability: the kappa statistic, Biochem Med 22 (3) (2012) 276–282.