Simple, Fast and Accurate Hyper-parameter Tuning in Gaussian-kernel SVM

Guangliang Chen
Department of Mathematics & Statistics
San José State University
San José, CA 95192–0103
guangliang.chen@sjsu.edu

Wilson Florero-Salinas
Mathematics Department
Foothill College
Los Altos Hills, CA 94022
florerosalinaswilson@fhda.edu

Dan Li
Department of Mathematics & Statistics
San José State University
San José, CA 95192–0103
lidan4201@gmail.com

***Abstract*—We consider the parameter tuning problem for Gaussian-kernel support vector machines, i.e., how to set its two hyperparameters – σ (bandwidth) and C (tradeoff). Among the many methods in the literature, the majority handle this task by maximizing the cross validation accuracy over the first quadrant of the (σ, C) plane. However, they are all computationally expensive because the objective function has no explicit formula so that one has to resort to numerical methods (which require training and testing the classifier many times). Additionally, these methods ignore the intrinsic geometry of training data and always operate in a large set, thus being computationally inefficient. In this paper we propose a two-step procedure for efficient parameter selection: First, we use a nearest neighbor method to directly set the value of σ based on the data geometry; afterwards, for the tradeoff parameter C we employ an elbow method that finds the smallest C leading to "nearly" the highest validation accuracy. By slightly sacrificing the validation accuracy our method gains additional attractive properties such as (1) *faster training* (i.e., much less candidate points to be examined) and (2) *better generalizability* (due to larger class margins). We conduct extensive experiments to show that such a combination of simple techniques achieves excellent performance - the classification accuracy of our method is comparable to its competitors in most cases, but it is much faster.**

I. Introduction

Due to its great flexibility and superior performance, the Gaussian-kernel Support Vector Machine (GkSVM) [1], [2] is one of the most popular classifiers used by machine learners. In the binary setting where the training data $\mathbf{X} = [\mathbf{x}_1 \dots \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ has only two kinds of labels $y_i = \pm 1$, GkSVM first maps the training examples into some feature space $\mathcal{F}$ by using a function $\Phi : \mathbb{R}^d \mapsto \mathcal{F}$ that is defined by

$$\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}') = \kappa(\mathbf{x}, \mathbf{x}') := e^{-\|\mathbf{x}-\mathbf{x}'\|^2/(2\sigma^2)} = e^{-\gamma\|\mathbf{x}-\mathbf{x}'\|^2}, \quad \forall\ \mathbf{x}, \mathbf{x}' \in \mathbb{R}^d \quad (1)$$

in which $\|\cdot\|$ represents the ℓ_2 vector norm and γ, σ are fixed constants with $\gamma = 1/(2\sigma^2)$. Afterwards, it finds a maximum-margin separating hyperplane $\mathbf{w} \cdot \Phi(\mathbf{x}) + b = 0$ in the feature space via a quadratic program

$$\min_{\mathbf{w},b,\{\xi_i\}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{subject to} \quad y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0,\ \forall\, i. \quad (2)$$

Here, the first term measures the (inverse) margin between the two classes in the feature space $\mathcal{F}$, and the second term "counts" the number of feature points $\Phi(\mathbf{x}_i)$ not *in ideal locations*[1]. The constant $C > 0$ is a tradeoff parameter whose value must be carefully tuned. This formulation is often called the *primal* problem, to distinguish from the *Lagrange dual*:

$$\max_{\lambda_1,\dots,\lambda_n} \ \sum_i \lambda_i - \frac{1}{2}\sum_i\sum_j \lambda_i\lambda_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \quad \text{subject to} \quad 0 \leq \lambda_i \leq C,\ \forall\, i \text{ and } \sum_i \lambda_i y_i = 0. \quad (3)$$

The dual formulation shows that the GkSVM learning problem depends on the training data only through their dot products in the feature space and thus eliminates the need to explicitly use the feature map Φ (which is often very high dimensional, and here infinite-dimensional). For a thorough tutorial on SVM and its various formulations we refer the reader to [3].

When there are more than two classes, a multiclass extension of the SVM is needed. Two simple and commonly-used extensions are the one-vs-one and one-vs-rest multiclass SVMs (see e.g. [4]). The former trains an SVM model between every pair of classes while the latter considers every class against all the other classes. Classification of a new instance for the two methods is done by assigning it to the "winning" class.[2] There are also other extensions which directly use a multiclass loss [5]. In this paper we focus on the one-vs-one extension (however, the techniques to be proposed later are independent of the multiclass extension used).

A practical difficulty with the GkSVM, however, is to set the two parameters γ and C which crucially affect the predictive performance. The majority of the existing methods in the literature handle this task solely as an optimization problem and try the best to achieve the highest cross validation (CV) accuracy:

$$\max_{\gamma>0,\, C>0} \ \mathrm{CVaccuracy}(\gamma, C). \quad (4)$$

[1] That is, they lie within the margin or on the wrong side of the hyperplane. In these cases, the slack variables ξ_i are positive; otherwise, they are zero.

[2] For one-vs-one multiclass SVM, the winning class is the most frequent prediction of the binary models, while for one-vs-rest multiclass SVM it is the class with the strongest positive prediction.

For example, the popular Grid Search algorithm – LIBSVM [4], [6] – selects the best pair out of a fixed grid in a large domain of the (γ, C) space, while Random Search [7], [8] uses randomly selected points from the domain (still a considerable fraction of the grid in size). Two other examples are Simulated Annealing [9], [10] and Bayesian optimization [11], [12] which employ sophisticated global optimization techniques to obtain optimal parameters. The former is a probabilistic technique for approximating the global optimum of a given function by simulating the cooling of material in a heat bath while the latter models a learning algorithm's generalization performance as a sample of the Gaussian process.

While some successes have been achieved by those methods, they suffer from the following drawbacks. First of all, all those methods operate in the entire first quadrant of the (γ, C) space and do not exploit the geometry of the training data to narrow down the search range for either of the two hyperparameters. Secondly, they all ignore the interpretation of the two hyperparameters - σ (scale) and C (margin) - and simply seek the highest possible CV accuracy. We point out that this is not the best strategy as it may be prone to overfitting and thus does not necessarily guarantee the highest test accuracy (see Fig. 1 for two examples). Finally and most importantly, these methods are all computationally very costly which thus may limit their use in practice.

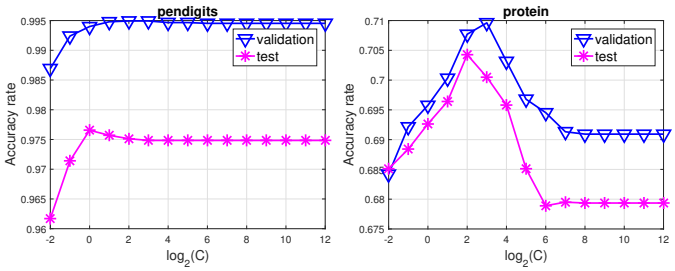

Fig. 1. Plots of the validation and test accuracy rates against different values of C for two data sets (γ is fixed in each plot). Observe that in both cases the C value leading to the highest validation accuracy does not lead to the highest test accuracy, but a preceding C value with comparable validation accuracy actually works better. This is because C controls the margin size of the trained SVM model; the smaller the parameter C, the larger the margin. With comparable validation accuracy rates, smaller C values thus lead to more robust models.

In this paper we address those issues and propose a two-step procedure for setting the two parameters:

(1) *Setting σ directly*. We point out that σ is a scale parameter which determines the smoothness of the GkSVM decision boundary and that its optimal value must reflect the local geometry of the training data. In addition, GkSVM is often more sensitive to σ than to C. Accordingly, we estimate the optimal value of σ directly from the training data by using the average distance of the training points to their nearest neighbors in the same class. This immediately reduces the two dimensional search domain to a one-dimensional set.

(2) *Picking C from a grid*. Once σ has been fixed, we will tune the C parameter in some finite grid. We have observed in many cases that the CV accuracy either peaks at, or stabilizes after, some C (as C increases); see again Fig. 1. This motivated us to propose a procedure that starts from the lower end of the C grid and gradually computes the validation accuracy curve, until an *elbow* point has been found (i.e., a point after which the validation accuracy practically stops increasing). Such a choice of C leads to bigger margins (better generalizability) and avoids testing the subsequent C values in the grid (faster training).

Though the combined algorithm only aims to achieve a comparable validation accuracy, it yields more robust models while checking significantly fewer candidate points (see Fig. 2, left diagram for a graphical comparison with grid search).

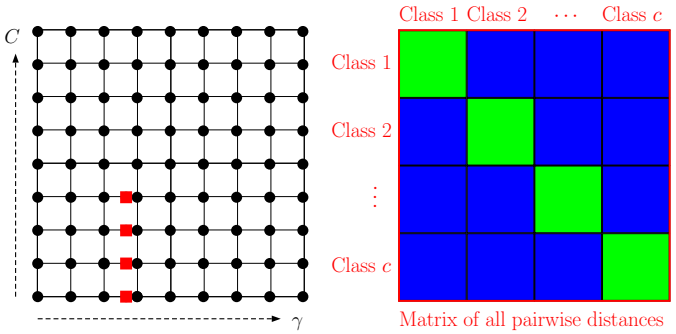

Fig. 2. Comparisons between the proposed method and existing techniques such as grid search (left diagram) and the Caputo and Jaakkola heuristics (right diagram). Left: The black dots represent the (γ, C) values examined by grid search while the red squares symbolize what our method tests. Right: The whole square (in red) represents the matrix of pairwise distances between all training points which have been sorted according to the labels. The Caputo heuristic uses certain percentile of these pairwise distances for estimating γ while Jaakkola is based on the pairwise distances between different classes (represented by the blue blocks). In contrast, our method uses only the distances that are between points in the same class (represented by the green blocks).

Our σ tuning method has a close connection to two heuristics in the literature, Caputo [13] and Jaakkola [14], which also try to learn the parameter directly from training data. The Caputo heuristic uses certain percentile (e.g., 10th, 50th and 90th) of all the pairwise distances among the training examples while Jaakkola uses the median value of the distances from each training point to the closest point from a different class. See Fig. 2, right diagram for an illustration. Our method has several advantages over these two heuristics. First, our method has a clear interpretation as it implies a nearest neighbor graph on the training data which is based on its local geometry. Second, our method only depends on the pairwise distances between points in the same class, and can be implemented very efficiently by using fast nearest neighbor search algorithms such as ANN [15]. Lastly, those two methods do not address how to tune the hyperparameter C.

The rest of the paper is organized as follows. In Section II we present our approach to parameter selection in GkSVM. Numerical experiments are performed in Section III to test the resulting algorithms. Finally, in Section IV, we conclude the paper while pointing out some future directions.

II. Proposed Methods

In this section we present our approaches to efficient and accurate selection of the parameters σ and C of the Gaussian-kernel SVM.

A. A kNN approach for directly setting σ

The kernel matrix $\mathbf{K}_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$ has different interpretations in different fields. In classification (which is the subject of this paper) it represents the dot product matrix between the images of the given training examples in a high dimensional feature space. In clustering, especially spectral clustering [16], $\mathbf{K}$ is understood as a similarity matrix defined on the given data (with larger values meaning more similar). Regardless of the interpretation, the common goal is to make, by varying σ, $\mathbf{K}$ as close to being block diagonal as possible (with the blocks corresponding to the classes). We adopt the spectral clustering perspective here to help select the optimal value of σ.

Ideally, all points in the same class should be assigned large similarity scores between them and those in different classes be given small similarity scores. However, for many practical data sets the classes often have complex shapes (e.g., nonconvex). Thus, with a single σ, at best one can only expect to assign large similarity scores to nearby points in each class. The bandwidth parameter σ plays an important role as it determines how fast the similarity score decays around each point. If σ is too small, then the similarity score decays too rapidly and consequently all points in each of the training classes (except for extremely close points) have small similarity scores between them. On the other hand, if σ is too large, then the similarity score decays too slowly and training examples from different classes may have large similarity scores which will complicate the subsequent classification task. Therefore, a tradeoff must be made when setting the value of σ (see Fig. 3 for a demonstration).

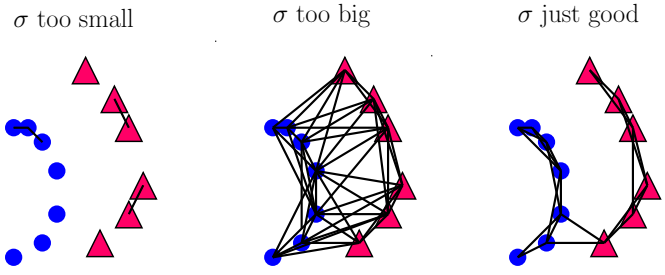

Fig. 3. Demonstration of the effect of σ on the structure of the kernel matrix, shown as a graph with nodes representing the training points and thickness of edges proportional to the corresponding entries of the kernel matrix (here only the thickest edges are displayed).

The above observations motivated us to adopt a nearest neighbor approach for directly setting the value of σ. To explain our method clearly, we introduce some notation first. Assume without loss of generality that $\mathbf{X} = [\mathbf{X}_1 \mid \cdots \mid \mathbf{X}_c] \in \mathbb{R}^{d\times n}$, where each $\mathbf{X}_\ell, 1 \le \ell \le c$ represents a class in the training set $\mathbf{X}$. Denote the size of $\mathbf{X}_\ell$ by n_ℓ. For every $1 \le i \le n$, let $\ell(i)$ represent the label of the class containing the training point $\mathbf{x}_i$, that is, $\mathbf{x}_i \in \mathbf{X}_{\ell(i)}$. We fix a positive integer k (typically between 6 and 10) representing the number of nearest neighbors to be examined (for each training vector). For any training example $\mathbf{x}_i$ we denote its kth nearest neighbor (kNN) in $\mathbf{X}_{\ell(i)}$ by $\mathbf{x}_i^{(k\text{nn})}$.

For an arbitrary training point $\mathbf{x}_i \in \mathbf{X}_\ell$, the distance from the point to its kth nearest neighbor in the same class $\mathbf{x}_i^{(k\text{nn})}$, i.e., $\|\mathbf{x}_i - \mathbf{x}_i^{(k\text{nn})}\|$, can be used as an estimate of the local scale of the class. We expect a considerable fraction of all the training samples in $\mathbf{X}$ that fall within this radius around $\mathbf{x}_i$ to still belong to $\mathbf{X}_\ell$. Since this estimate may slightly vary from location to location, we use all training points of $\mathbf{X}_\ell$ to obtain a robust estimate of its local scale:

$$\tilde{\sigma}_\ell = \frac{1}{n_\ell} \sum_{\mathbf{x}_i \in \mathbf{X}_\ell} \|\mathbf{x}_i - \mathbf{x}_i^{(k\text{nn})}\|. \tag{5}$$

We then set σ as a weighted average of the class scales

$$\tilde{\sigma} = \frac{1}{n} \sum_{\ell=1}^{c} n_\ell \tilde{\sigma}_\ell. \tag{6}$$

This is equivalent to calculating the global average of the distances between all training points and their respective kNNs within the same class:

$$\tilde{\sigma} = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathbf{x}_i^{(k\text{nn})}\|. \tag{7}$$

In [17] we used the same formula for estimating σ but $\mathbf{x}_i^{(k\text{nn})}$ was defined with respect to the full training set $\mathbf{X}$ rather than the associated class $\mathbf{X}_{\ell(i)}$.

When the data set is large (in size or dimension or both), the nearest neighbor search for all points in their associated training classes may become a computational burden. To increase efficiency, we propose a *stochastic* kNN method to estimate the deterministic quantity $\tilde{\sigma}$ by using only a small stratified sample of s training vectors $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_s}$ from $\mathbf{X}$: For each $1 \le \ell \le c$, we select at random $s_\ell = \lceil s \cdot n_\ell/n \rceil$ points from $\mathbf{X}_\ell$ proportionally to its size and set σ as

$$\tilde{\sigma}^{(s)} = \frac{1}{s} \sum_{j=1}^{s} \|\mathbf{x}_{i_j} - \mathbf{x}_{i_j}^{(k\text{nn})}\|. \tag{8}$$

It is easy to see that this is also a weighted average of the scales of the different classes which are estimated from a small number of samples in each class.

Typically, we choose $s = 50$ due to the Law of Large Numbers (LLN). Experiments conducted later in this paper also examine other values of s and indicate that a sample size as small as 10 may be already enough in many cases.

B. An elbow method for tuning C

The tradeoff parameter C also plays an important role in the training of a Gaussian-kernel SVM model. Generally, it is inversely related to the identified margin of the model: Large values of C force SVM to penalize heavily on the number of training errors and thus would lead to small margins between the different training classes, with a risk of overfitting the data. On the other hand, small values of C can tolerate many training errors and lead to large margins which could underfit the data. Therefore, caution has to be used when selecting the optimal value of C and a tradeoff must be reached between the number of training errors and the size of the margin.

With the introduction of the kNN tuning method for direct selection of σ (and γ) in the preceding section, one can then perform grid search solely for C (based on the fixed σ), that is, to examine all values of C from a large candidate set and choose the one that maximizes the cross validation accuracy. While grid search guarantees that numerically the highest cross validation accuracy be achieved (over the given candidate set), we actually prefer smaller values of C that lead to very close validation accuracy rates. The reason is that smaller values of C imply bigger margins and thus produce more robust models that may generalize better to previously unseen data (see Fig. 1 for examples). Therefore, we propose to use the smallest C that leads to a comparable validation accuracy, which often appears as an elbow point on the validation accuracy curve. Specifically, we start from the smallest value of a chosen C grid and gradually compute the validation accuracy curve. We will stop as soon as an elbow point is detected. Such a procedure avoids testing larger values of C and thus can significantly reduce the amount of training time.

C. The combined algorithm

We combine the two parameter tuning techniques presented earlier to form a new algorithm for hyperparameter tuning in GkSVM (see Alg. 1). We note that in the elbow search step of Alg. 1 we declare a C value an elbow point if the validation accuracy for the first time does not increase more than a chosen tolerance $\epsilon > 0$ at the next two C values. In all experiments of this paper we fix $\epsilon = 0.005$ (i.e., 0.5%).

Algorithm 1 Proposed algorithm: kNN-GkSVM

Input: Training set $\mathbf{X}$, # nearest neighbors k, sample size s, # CV folds f, and bounds $C_{\min}, C_{\max}$ for C

Output: Optimal σ, C as well as the trained Gaussian-kernel SVM model

Steps:

1: **kNN tuning for** σ:

- Sample s points from $\mathbf{X}$, with the number of samples from each training class proportional to the size of the class
- For each of the s points sampled, find the kth nearest neighbor of that point in the class it belongs to.
- Use Eq. (8) to set σ and fix the value for below.

2: **Elbow search for** C (based on the σ obtained above):

- Randomly partition $\mathbf{X}$ into f equal subsets, and fix it for the steps below.
- *Initialize* $C = C_{\min}$. Cross validate this choice of C and the σ from Step 1 based on the fixed partition. Compute the corresponding validation accuracy.
- *Repeat*
  - Increase C by doubling it: $C \longleftarrow C \cdot 2$
  - Cross validate the new C and the same σ from above based on the fixed partition of $\mathbf{X}$ to get a new validation accuracy.

  Until an elbow C has been found, or $C = C_{\max}$.

TABLE I
SUMMARY INFORMATION OF THE DATA SETS USED.

dataset	source	#classes	#dims	#train	#test
astroparticle	[19]	2	4	3089	4000
bioinformatics	[19]	2	21	1243	41
dna	Statlog [20]	3	180	2000	1186
letter	Statlog	26	16	15000	5000
madelon	[21]	2	500	2000	600
pendigits	UCI [22]	10	16	7494	3498
protein	[23]	3	357	17766	6621
satimage	Statlog	6	36	4435	2000
shuttle	Statlog	7	9	43500	14500
splice	UCI	2	60	1000	2175
svmguide4	[19]	6	10	300	312
usps	[24]	10	256	7291	2007
vowels	UCI	11	10	528	462

III. EXPERIMENTS

In this section we conduct extensive numerical experiments to examine both the accuracy and speed of Alg. 1 relative to existing parameter selection methods, namely *Grid Search* [6], *Random Search* [7], *Caputo Heuristic* [13], *Jaakkola Heuristic* [14], *Simulated Annealing* [18], and *Bayesian* [11].

A. Data sets and experimental setup

We choose 13 benchmark data sets from the LIBSVM website at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ for our study; summary information of the data sets along with their sources is displayed in Table I. These data sets are already partitioned into training and test parts, which allows for easy reproducibility of the results reported in this paper.

Following the recommendation in [4] we rescaled all features of the 13 data sets to the $[0, 1]$ range (though this is not necessary for our proposed methods). We kept all the features and did not reduce the dimensionality of the data. For the data sets with more than two classes, we adopted the one-vs-one multiclass extension of the binary GkSVM classifier. The number of cross-validation folds was set to 10 in all experiments.

To make the parameter tuning problem (4) practically tractable and for the reason of fair comparison we followed [4] to restrict all methods to work in the same (γ, C) domain $\Omega = [2^{-10}, 2^4] \times [2^{-2}, 2^{12}] \subset \mathbb{R}^2$. All experiments were conducted on a compute server with 48 GB RAM and 2

CPU's with 12 total cores. We recorded the CPU time for each experiment.

We implemented all methods except Bayesian in MATLAB 2016a based on its Statistics and Machine Learning Toolbox (mainly the two functions *fitcsvm* and *predict*). For Bayesian optimization, we used Python's SPEARMINT package at https://github.com/HIPS/Spearmint for convenience (we also coded up Deterministic kNN in Python in order to compare the running time of the two algorithms). The parameters uniquely associated to each algorithm are described below:

- **Grid Search**: We used the following candidate sets from [4] for γ and C respectively: $\gamma \in \{2^{-10}, 2^{-9}, \ldots, 2^{4}\}$ and $C \in \{2^{-2}, 2^{-1}, \ldots, 2^{12}\}$. We also used the same C grid for Caputo, Jaakkola, Deterministic and Stochastic kNN which all perform grid search for the parameter C.
- **Random Search**: We selected 60 points at random (in $\log_2$ scale) from the fixed region Ω. Due to the randomness in sampling we repeated random search for each data set 10 times to compute the average test accuracy and CPU time.
- **Caputo**: We tried each of the 10th, 50th and 90th percentiles of all pairwise distances for σ and found that the 10th percentile consistently performed the best, so we only report the results with this percentile.
- **Jaakkola**: We set σ equal to the median Euclidean distance between all training vectors and their closest neighbors from a different class.
- **Simulated Annealing (SA)**: We employed *hide and seek SA* [18] which performs an immediate annealing whenever a better solution is found rather than waiting for the current annealing cycle to end, in order to converge more quickly to the global optimum. We used the recommended parameter values in [18]. Like random search, we performed 10 independent trials for each data set to report the average test accuracy and CPU running time.
- **Bayesian**: We applied SPEARMINT with 50 iterations.
- **Deterministic kNN**: The number of nearest neighbors used for computing σ was fixed to $k = 7$ for all data sets. To perform nearest neighbor search we simply used the MATLAB 2016a *knnsearch* function.
- **Stochastic kNN**: We used the same options for deterministic kNN and additionally set $s = 50$ (number of samples used) in all cases. Similarly to random search and simulated annealing, we repeated stochastic kNN for each data set 10 times to deal with randomness.

The test accuracy rates obtained by the different methods are summarized in Table II. The following observations are at hand: (1) Our methods achieve the highest test accuracy rates (those **highlighted**), or stay within a 1% absolute difference from the best ones (those underlined), on all but two data sets ('madelon' and 'svmguide4'); (2) Between deterministic and stochastic kNNs, the two of them are less than 0.5% apart in all cases except for 'bioinformatics' (however, the difference is still less than 2%). Overall, our methods obtained very competitive accuracy rates.

TABLE II
ACCURACY RATES (%) OBTAINED BY GRID SEARCH (GS), RANDOM SEARCH (RS), CAPUTO (CP), JAAKKOLA (JK), SIMULATED ANNEALING (SA), BAYESIAN (BY), DETERMINISTIC kNN (DkNN) AND STOCHASTIC kNN (SkNN). THOSE **HIGHLIGHTED** NUMBERS ARE THE HIGHEST ACCURACY ACHIEVED ON EACH DATA SET. FOR CONVENIENCE, WE ALSO UNDERLINE THE ACCURACY RATES ACHIEVED BY OUR METHODS WHEN THEY ARE LESS THAN 1% FROM THE BEST RATES.

dataset	GS	RS	CP	JK	SA	BY	DkNN	SkNN
astroparticle	96.2	96.5	96.8	96.8	96.5	**97.2**	96.5	96.3
bioinformatics	85.4	85.8	**95.1**	**95.1**	83.9	85.4	**95.1**	93.4
dna	93.9	94.8	**95.6**	95.3	95.0	95.0	95.3	95.3
letter	97.9	97.9	97.6	97.9	97.9	97.9	**98.0**	**98.0**
madelon	**61.2**	59.7	59.0	59.5	59.3	59.2	59.7	59.7
pendigits	98.3	98.2	**98.5**	98.3	98.2	**98.5**	97.7	97.6
protein	69.6	70.2	70.3	**70.4**	**70.4**	70.3	70.1	70.2
satimage	90.3	90.6	90.3	90.5	90.5	**90.9**	90.6	**90.9**
shuttle	**99.9**	**99.9**	**99.9**	**99.9**	**99.9**	**99.9**	99.4	99.8
splice	89.7	89.8	90.2	89.7	90.0	**90.3**	90.1	90.1
svmguide4	**88.5**	88.1	73.7	68.3	88.0	87.5	77.2	77.0
usps	95.3	95.3	95.3	95.2	95.3	**95.5**	95.4	95.3
vowels	61.7	62.6	**64.9**	63.6	48.9	62.3	64.5	64.3

The CPU time taken by the different methods is shown in Figs. 4 (MATLAB implementations of all methods except Bayesian) and 5 (Python implementations of deterministic kNN and Bayesian). We see that stochastic kNN is consistently the fastest method (the deterministic method is close except for 'shuttle') while simulated annealing, grid search and Bayesian are the slowest methods (about two magnitudes slower than stochastic kNN in all cases). We further plot in Fig. 6 the elbow C values detected by the deterministic method on all data sets. They seem to be around the value 2^2 which implies that on average only 5 values in the C grid (plus a couple extra ones needed for elbow detection) were actually tested by our method; this is in contrast to all the other methods which exhaustively test all 15 values. An implementation of the elbow C method for Jaakkola obtained similar efficiency while maintaining comparable accuracy (see Fig. 7).

Lastly, we test the stability of our methods in regards to the two parameters k, s. In Fig. 8, we applied the deterministic method with $k = 1, 2, \ldots, 12$ to all data sets and display the corresponding accuracy curves (as functions of k). We see that all the curves stabilize after $k = 6$, showing that the deterministic method is insensitive to the choice of k (as long as $7 \leq k \leq 12$). In Fig. 9, we fixed $k = 7$ for stochastic kNN but set $s =$ 10, 20, 30, 40, 50, 75, 100 respectively to study the sensitivity of this parameter. Due to the randomness in the subset selected for estimating σ we conducted 10 independent trials for each value of s and took the average of the accuracy results. We can see that like the parameter k, the choice of s is insensitive either and an s value as small as 10 already gives very close results to the deterministic method.

IV. CONCLUSIONS AND FUTURE WORK

We presented simple yet effective machine learning techniques for efficiently tuning the two hyperparameters γ, C of the Gaussian-kernel SVM. Unlike the existing techniques

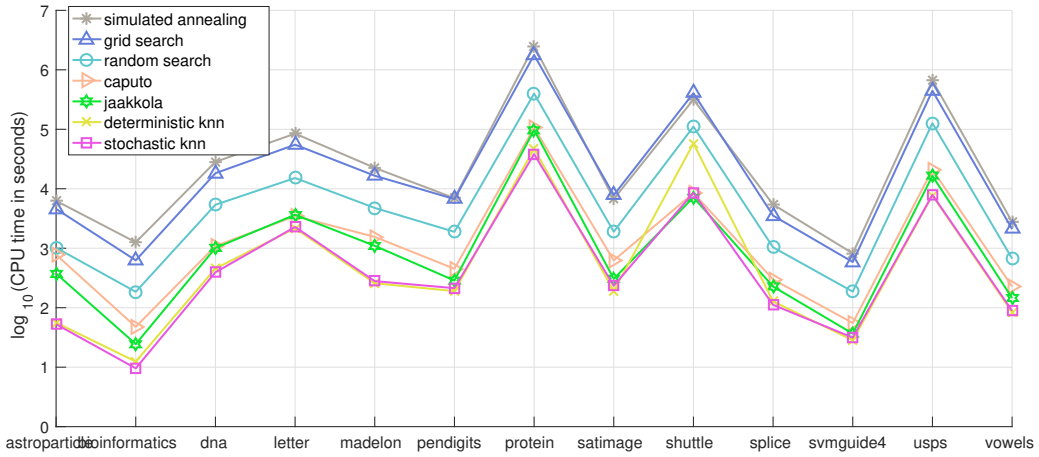

Fig. 4. CPU time of the different parameter selection methods (except Bayesian), all implemented in MATLAB.

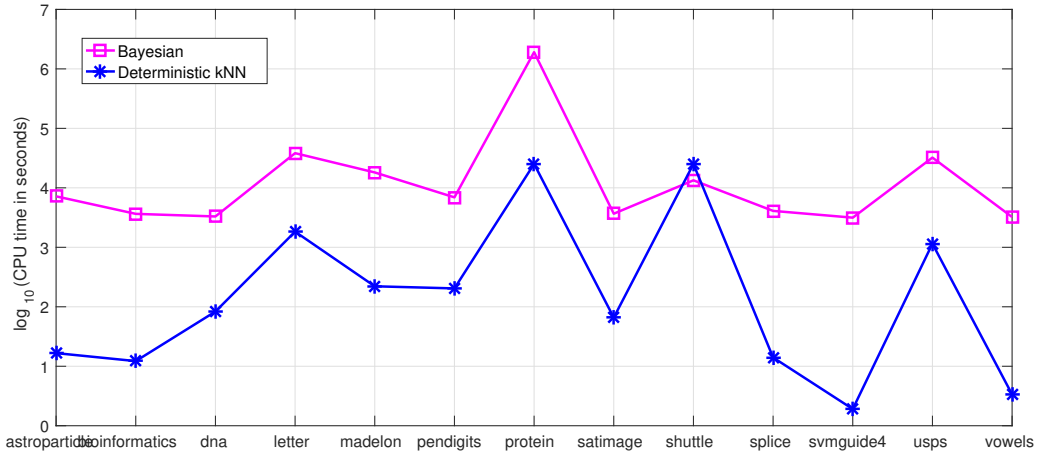

Fig. 5. CPU time of the deterministic kNN and Bayesian methods implemented in Python.

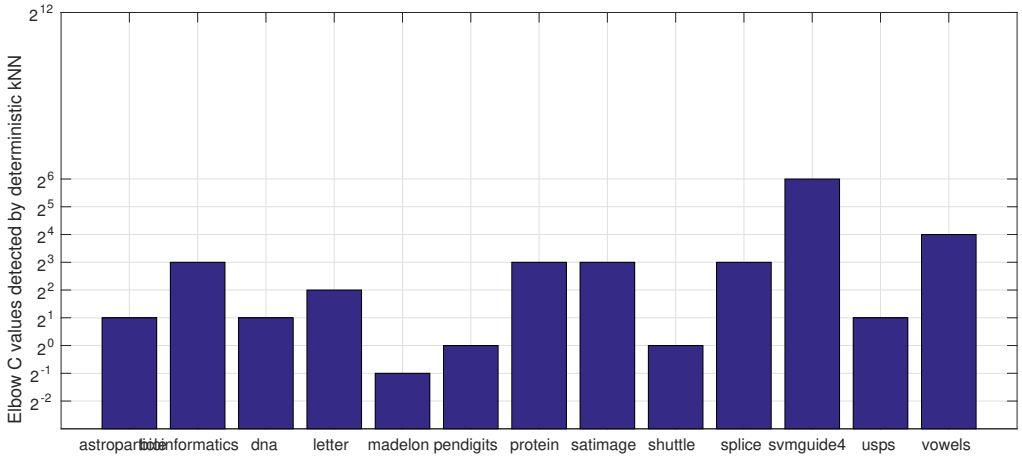

Fig. 6. Elbow C values detected by deterministic kNN on all 13 data sets.

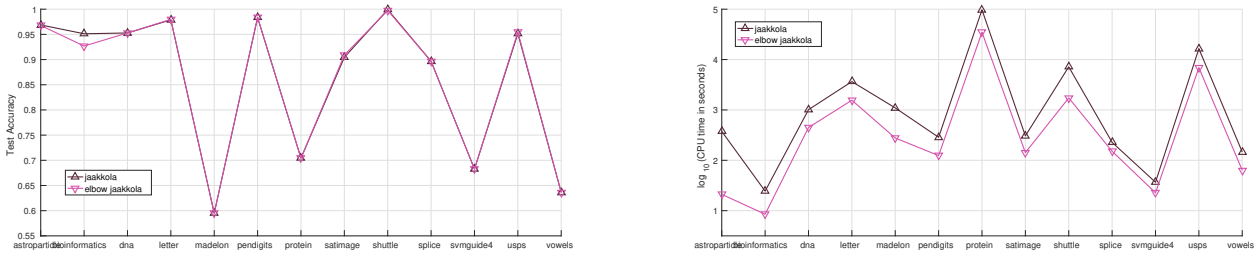

Fig. 7. We implemented the elbow C method together with the Jaakkola heuristic to compare with the plain Jaakkola method. Left and right plots respectively display the test accuracy rates and running times of both methods on the same 13 data sets we used earlier. It is clear that the elbow method considerably reduced the running time while maintaining the same test accuracy rates (except in one case).

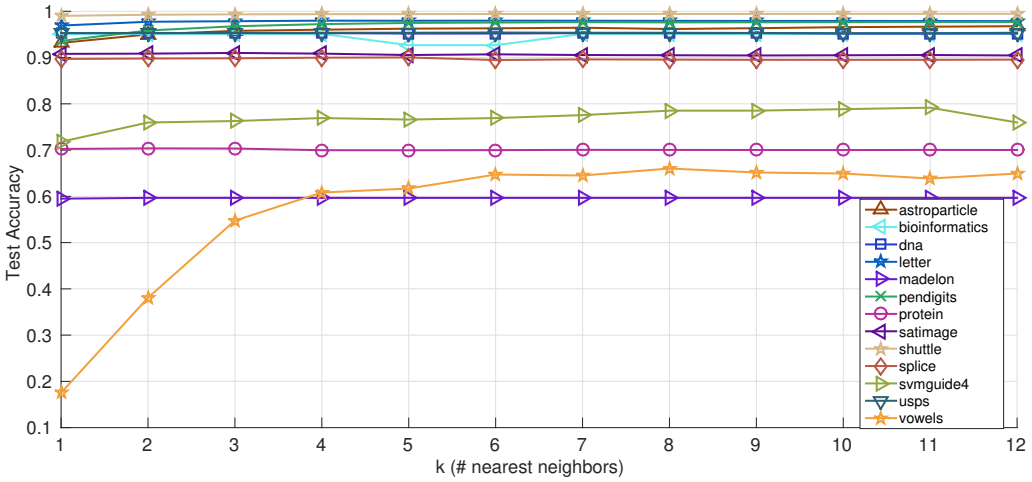

Fig. 8. Test accuracy rates obtained by the *deterministic kNN* method with different choices of k.

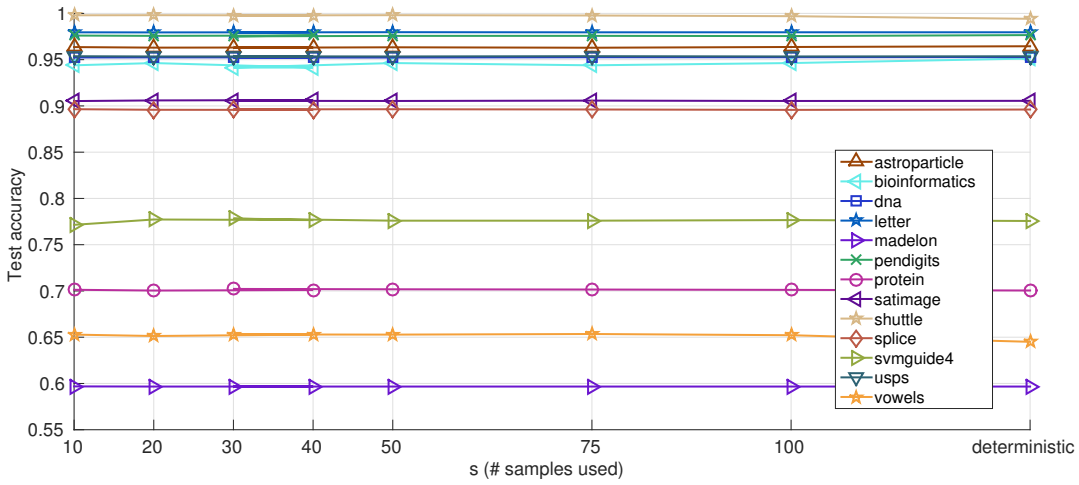

Fig. 9. Test accuracy rates obtained by the *stochastic kNN* method with different choices of s (k is fixed to 7 all the time). For easy comparison we have added the results of the deterministic method to the end of the curves.

which are all based on the formulation in (4), we exploit the geometry of the training data to select γ directly and take the margin size into consideration by choosing the elbow C that leads to comparable cross validation accuracy rather than spending expensive computational power in finding the highest validation accuracy. Our combined algorithm has clear interpretations and is expected to prevent overfitting to the training data. Meanwhile, it enjoys a fast speed by significantly reducing the number of candidate points to be checked while maintaining comparable accuracy, both demonstrated through extensive experiments. It has also been experimentally verified that our method is insensitive to choices of the two parameters used for calculating σ, namely, k (number of nearest neighbors) and s (number of samples). Overall, our method proves to be very robust and comparatively accurate, but significantly faster than its competitors.

We mention a few directions that we are currently working on or wish to explore in the near future. First of all, we would like to compare different distance metrics such as ℓ_p in terms of the prediction performance (as ℓ_2 is not necessarily the optimal choice) and find which one best adapts to the data geometry. Secondly, since the parameter σ of the Gaussian kernel is a global constant, it may not be able to capture the full complexity of real data sets which often exhibit different scales in different locations. We are currently working to address this issue. Thirdly, we point out that the elbow C method can also be used together with other parameter tuning methods for Gaussian-kernel SVM (e.g. Caputo) or even in other settings such as polynomial kernel SVM and neural networks to improve their speed and/or accuracy. We plan to explore this direction too. Lastly, we have observed that high dimensional data sets (such as 'protein' and 'usps') require significantly longer time to train a Gaussian-kernel SVM model. Additionally, it is known that Gaussian-kernel SVM is not better (and sometimes worse) than linear SVM when there are a large number of features [19]. We are currently experimenting with novel feature selection techniques to boost the performance of Gaussian-kernel SVM in those situations.

ACKNOWLEDGMENT

The authors thank the anonymous referees for their helpful feedback. G. Chen was supported by the Simons Foundation Collaboration Grant for Mathematicians while conducting the research presented in this paper. W. Florero-Salinas completed part of the work as a graduate student at San José State University. Finally, the authors gratefully acknowledge the Woodward Fund for Applied Mathematics at San José State University, a gift from the estate of Mrs. Marie Woodward in memory of her son, Henry Tynham Woodward, for generous support through sponsoring a directly-related student research program (which both W. Florero-Salinas and D. Li took part in) and providing a summer grant to G. Chen.

REFERENCES

[1] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *COLT '92 Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, 1992, pp. 144–152.

[2] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[3] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[4] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[5] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *The Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2002.

[6] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2011. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

[7] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012.

[8] R. Mantovani, A. Rossi, J. Vanschoren, B. Bischl, and A. de Carvalho, "Effectiveness of random search in SVM hyper-parameter tuning," in *Proceedings of 2015 International Joint Conference on Neural Networks*, pp. 1–8.

[9] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of Chemical Physics*, vol. 21, pp. 1087–1092, 1953.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[11] J. Snoek, H. Larochelle, and R. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 2951–2959.

[12] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010. [Online]. Available: https://arxiv.org/pdf/1012.2599.pdf

[13] B. Caputo, K. Sim, F. Furesjo, and A. Smola, "Appearance-based object recognition using SVMs: Which kernel should I use?" in *Proceedings of NIPS workshop on Statistical methods for Computational Experiments in Visual Processing and Computer Vision*. Whistler, 2002, pp. 149–158.

[14] T. Jaakkola, M. Diekhaus, and D. Haussler, "Using the fisher kernel method to detect remote protein homologies," in *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*. AAAI Press, 1999, pp. 149–158.

[15] D. Mount and S. Arya, "ANN: A library for approximate nearest neighbor searching," 2010. [Online]. Available: https://www.cs.umd.edu/~mount/ANN/

[16] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Advances in Neural Information Processing Systems*, vol. 14, 2001, pp. 849–856.

[17] W. Florero-Salinas, D. Li, and G. Chen, "A nearest neighbor approach for efficient selection of the bandwidth parameter in gaussian-kernel support vector machines," in *Proceedings of 2016 Pacific Conference on Statistical Computing and Data Mining*. The Sisu Advantage, 2016, pp. 25–34. [Online]. Available: http://www.thinksisu.org/event/statistical-computing/#proceedings

[18] S.-W. Lin, Z.-J. Lee, S.-C. Chen, and T.-Y. Tseng, "Parameter determination of support vector machine and feature selection using simulated annealing approach," *Appl. Soft Comput.*, vol. 8, no. 4, pp. 1505–1512, 2008.

[19] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," 2003, technical report, Department of Computer Science, National Taiwan University. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

[20] B. Henery, "The Statlog collections," 1993. [Online]. Available: http://archive.ics.uci.edu/ml/machine-learning-databases/statlog/

[21] I. Guyon, S. Gunn, A. B. Hur, and G. Dror, "Result analysis of the NIPS 2003 feature selection challenge," in *Advances in Neural Information Processing Systems*, vol. 17, 2005.

[22] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[23] J.-Y. Wang, "Application of support vector machines in bioinformatics," 2002, master's thesis, Department of Computer Science and Information Engineering, National Taiwan University.

[24] J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.