Classification and Clustering using Generalized Support Vector Machines and Tuning Metaheuristics

Stephen Winters-Hilt[1,2,*]

[1] Dept. of Computer Science, University of New Orleans, 2000 Lakeshore Drive, New Orleans, LA 70148, USA
[2] Meta Logos Incorporated, 6218 Waldo Dr., New Orleans, LA 70122, USA

*email: winters@cs.uno.edu; winters@meta-logos.com

Abstract
Support Vector Machine (SVM) methods are described for data classification, data clustering, and signal analysis. The SVM implementations described involve SVM algorithmic variants, SVM kernel variants, and SVM chunking variants, as well as SVM tuning and clustering metaheuristics. The SVM discussion is interwoven with application to efforts in biophysics, with application to the 'information flows' associated with the nanopore transduction detector (NTD) in particular. Analysis of the NTD signal is done using a generalized set of hidden Markov model (HMM) and Support Vector Machine (SVM) based tools, the latter described here, as well as metaheuristics for tuning and selection.

SVM training metaheuristics are also described that make use of the SVM's confidence parameter to bootstrap from a strong classification engine to a strong clustering engine via use of label changes, and repeated SVM training processes with the new label information obtained.

Introduction

A classifier is typically a simple rule whereby a class determination can be made, such as a decision boundary (see Fig. 1). Learning the decision rule, or a sufficiently good decision rule, especially if simple and elegant, is the implementation aspect of a classifier, and can be difficult and time consuming. Even so, this is often manageable because at least there is data to 'learn from', e.g., supervised learning, with instances and their classifications (or 'labels'). Learning for classification can be done very effectively using generalized Support Vector Machines (SVMs), as will be described in what follows. With clustering efforts, or unsupervised learning, on the other hand, we don't have the label information during training. In what follows SVMs will also be shown to be incredibly effective at clustering when used with metaheuristics to recover label information in a bootstrap learning process. Also shown will be implementation details for distributed SVM training, and other speedup optimizations, for practical deployment of the generalized SVM classification and clustering methods in real-time operational situations (as demonstrated in analysis in nanopore detector experiments).

The SVM-based clustering method that will be described here makes use of the SVM-classifier convergence process. Single-convergence initialized clustering methods, involving label-flipping between SVM convergence training runs, have been studied previously and will be described in the Background Section. The single-convergence methods outperform other methods on the test sets considered, but in examining the clustering failures (albeit fewer than with parameterized methods), there appears to be room for improvement. Efforts to handle this with more sophisticated tuning have met with initial success, as will be related in the Results. Before proceeding with further discussion along these lines, however, there is one other form of SVM-based clustering initialization to mention that may be adopted and modify the overall approach, in many cases with significant improvement overall, as well as reduction in complexity. That different approach is to initialize with information from *multiple* SVM convergences, with selection on training data based on algorithmic methods that leverage the clustering groupings indicated. This can be done to more effectively cluster, or cluster with less sophistication, and initial efforts along these lines as described in the Discussion section.

In setting up an SVM Classifier, one must have training data in the form of feature vectors, where all of the feature vectors are the same length. One typically needs to specify a choice of kernel and kernel parameter (and possibly other parameters), and therein lies the rub. The SVM may not converge with your specification. SVMs have a surprising amount of practical functionality, however, as will be shown. It is fairly easy to tune SVMs, in many cases, by simply using a default set of kernel's and parameter ranges. There is more robust performance, however, with more sophisticated tuning. In the SVM applications to bootstrap a clustering solution, there is more sensitivity to kernel and kernel parameter overall, and more sophisticated tuning methods are clearly helpful, as will be described further in the Discussion.

Use of SVMs for clustering (unsupervised learning) is possible in a number of different ways. As with the multiclass SVM discriminator generalizations, the strong performance of the binary SVM enables SVM-External as well as SVM-Internal approaches to clustering. Non-parametric SVM-based clustering methods may allow for much improved performance over parametric approaches, particularly since they can apparently be designed to inherit the strengths of their supervised SVM counterparts as will be shown. The 'external' SVM clustering algorithm,

described in detail in the Methods and Results, clusters data vectors with no *a priori* knowledge of each vector's class.

In application to channel current signal analysis, also briefly described in what follows, there is generally an abundance of experimental data available, if not, the experimenter can usually just take more samples and make it so. In this situation it is appropriate to seek a method good at both classifying data and evaluating a confidence in the classifications given. In this way, data that is low confidence can simply be dropped. The structural risk minimization at the heart of the SVM method's robustness *also provides a strong confidence measure*. For this reason, SVM's are the classification method of choice for channel current analysis, as they have excellent performance at 0% data drop, and as weak data is allowed to be dropped, the SVM-based approaches far exceed the performance of most other methods.

The ability to do fast SVM training (with distributed chunking and, possibly, GPU enhancements) means that ***online*** SVM learning can be managed in a brute-force fashion, with re-tuning on kernels periodically, and directly re-training on a moving window of data. Note: the applications of the SVM methods not only include classification and clustering, but also impact feature extraction and identification in HMM-based methods, using an HMM/SVM vectorization/classification boost [1], among other things.

Background

In Fig. 1 a decision boundary is shown as the solid line for a 2-D domain, where separating hyperplane generalizations to planes in 3-D and hyperplanes in higher dimensional domains are also possible (within any orientable hyperplane manifold). The notion of a separating hyperplane is not unique to the SVM approach, but it is with use of further Structural Risk Minimization (SRM) constraints via maximizing a margin around the decision hyperplane, where there are constrained to be low numbers of training instances in the margin region (zero if fully separable). The margin is shown as the region around the decision boundary in Fig. 1 that is between the dotted lines on either side of the decision hypersurface. Generalizations to compact or multiple decisions surfaces are also possible. In the case of the circle, this provides a method for (SVM 'Internal') clustering [29].

Kernel modeling

The so-called curse of dimensionality from statistics says that the difficulty of an estimation problem increases drastically with the dimension N of the space, since in principle as N increases, the number of required patterns to sample grows exponentially. This statement may cast doubts on using higher dimensional feature vectors as input to learning machines. This must be balanced with results from statistical learning theory [8], however, that show that the likelihood of data separability by linear learning machines is proportional to (and improves with) their dimensionality.

Thus, instead of working in the $\boldsymbol{R}^N$, one can design algorithms to work in feature space, $\boldsymbol{F}$, where the data has much higher dimension -- but with sufficiently small function class. This can be described via the following mapping

$$\Phi: \boldsymbol{R}^N \rightarrow \boldsymbol{F}\ ;\ \boldsymbol{x} \rightarrow \Phi\ (\boldsymbol{x}).$$

Consider the prior training description with data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \boldsymbol{R}^N$ is mapped into a potentially much higher dimensional feature space $\boldsymbol{F}$. For a given learning algorithm one now considers the same algorithm in $\boldsymbol{F}$ instead of $\boldsymbol{R}^N$. Hence, the learning machine works with the following:

$$(\Phi(\boldsymbol{x}_1), y_1), \ldots, (\Phi(\boldsymbol{x}_n), y_n) \in \boldsymbol{F} \times Y,\ Y = \{\pm 1\},$$

where the kernel is $\Phi(\boldsymbol{x}_i) \bullet \Phi(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j) = K_{ij}$. It is important to note that this mapping is also implicitly done for (one hidden layer) neural networks, radial basis networks [9] and boosting algorithms [10] where the input data is mapped to some representation given by the hidden layer, the radial basis function (RBF) bumps or the hypotheses space, respectively.

Kernel tuning

It is conceivable to have a properly coded SVM but to initiate SVM training with model parameters, such as the kernel or kernel parameter, that are so far out of the operational regime that no convergence is obtained in training. So training must be repeated, with tuning on SVM parameters, to optimize. For some feature vectors, such as probability vectors, this can partly be done automatically with choice of kernel. Overall, in many situations the SVM tuning can be done quickly, manually, and to some extent automatically, with simple range testing, where only small, separated, subsets of the training data are used in the tuning tests, before performing SVM training on the full dataset minus the tuning data. Sometimes more elaborate tuning procedures are needed, however, and thus necessary for performance guarantees, and also for the SVM applications in clustering that will be described in the Results and the Methods. Tuning is a form of optimization, and excellent metaheuristics are known for identifying optimal solutions when a scoring function (a fitness function) can be identified (such as for the SVM sensitivity and specificity score). Metaheuristics optimization includes genetic algorithms, simulated annealing, swarm intelligence, ACO, steepest ascent hill-climbing, among others. Applications of many of these methods are shown in the results involving SVM-external clustering.

Kernel construction using polarization

The kernels used in the analysis are based on a family of previously developed kernels [11,12], here referred to as 'Occam's Razor', or 'Razor' kernels. As will be seen, the Gaussian kernel is included in the family of Razor kernels. All of the Razor kernels examined perform strongly on the channel current data analyzed, with some regularly outperforming the Gaussian Kernel itself. The kernels fall into two classes: regularized distance (squared) kernels; and regularized information divergence kernels. The first set of kernels strongly models data with classic, geometric, attributes or interpretation. The second set of kernels is constrained to operate on $(\boldsymbol{R}^+)^N$, the feature space of positive, non-zero, real-valued feature vector components. The space of the latter kernels is often also restricted to feature vectors obeying an L_1-norm = 1 constraint, i.e., the feature vector is a discrete probability vector. For further details, see the Methods

Given any metric space (χ, d) one can build a positive-definite kernel of the form $e^{-\lambda d^2}$. Conversely, any positive definite kernel with form $e^{-\lambda d^2}$ must have a 'd' that is a metric (this is Mercer's condition in another form). (It is important to note that the metric appears as a squared entity in the argument, which is a mathematical entity that must, at least, satisfy the triangle inequality. It so happens that there ia another set of entities, other than metrics squared, that satisfy the triangle inequality, and these relate to the Bregman divergences, and the information

divergences such as relative entropy, in particular, where relative entropy is a fundamental (simple) divergence, just as Euclideane distance is a fundamental (simple) metric. This suggests that the 'simplest' kernel is the Gaussian kernel, since the 'simplest' distance, the Euclidean distance, is used. Likewise, this suggests that the simplest that the simplest divergence-based kernel would be the aforementioned entropic kernel.

The use of probability vectors, and L_1-norm feature vectors in general (often in conjunction with the entropic kernel), turns out to be a very general formulation, wherein feature extraction makes use of signal decomposition into a complete set of separable states that can be interpreted or represented as a probability vector (or normalized collection of such, or concatenation, then normalization, etc.). A probability vector formulation also provides a straightforward hand-off to the SVM classifiers since all feature vectors have the same length with such an approach. What this means for the SVM, however, is that geometric notions of distance are no longer the best measure for comparing feature vectors. For probability vectors (i.e., discrete distributions), the best measures of similarity are the various information-theoretic divergences: Kullback–Leibler, Renyi, etc. By symmetrizing over the arguments of those divergences, the entropic kernels are obtained, where the (symmetrized) Kullbach-Leibler Diveregence is used in the entropic kernel in [12-14] and in the Results that follow).

Methods

SVM Lagrangian formulation

The SVM approach encapsulates a key Structured Risk Minimization (SRM) criterion when it seeks to obtain the separable solution for which "d" is the greatest. This is the solution for which the separating hyperplane is the furthest distance possible from the positive & negative support vectors (the nearest data points if separable), which permits a structured risk minimization.

In order to formulate the SVM Lagrangian, we first need multipliers for the collection of separability constraints on solutions: $y_i(\omega \bullet \boldsymbol{x} - b) - 1 \geq 0 \ \forall i$. For SRM, we then need to maximize $d=2/\|\omega\|$, or minimize $\|\omega\|^2$, which is chosen due to simplifications in the formalism that follows (i.e., if we max $2/\|\omega|$by min on $\|\omega|$, it could just as well be done with min on $\|\omega\|^2$). The Lagrangian formulation then should have one multiplier constraint for each training instance, where $y_i(\omega \bullet \boldsymbol{x} - b) - 1 \geq 0$, and minimize on $\|\omega\|^2$ overall, so:

$$L(\vec{\omega}, b, \vec{\alpha}) = \frac{1}{2}\|\omega\|^2 - \sum_i \alpha_i [y_i(\omega x_i - b) - 1], \alpha_i \geq 0$$

We seek to minimize L on $\{\vec{\omega}, b\}$ and to extremize (maximize in this case) L on $\{\vec{\alpha}\}$, i.e., we seek a minimization -- maximization saddle-point optimization for the solution.

The Wolfe Dual Calculations, with or without slack variable, have the form:

$$\widetilde{L}(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j, \ \alpha_i \geq 0$$

where we want to find the $\alpha^{'s}$ that maximize $L(\alpha)$. Similarly, for the L_σ Dual:

$$\widetilde{L}_\sigma(\alpha) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j, \ C \geq \alpha_i \geq 0$$

So, the duals are the same aside from the $C \geq \alpha_i$ ($\max(\alpha) \leq C$) constraint.

SVM Kernels and Algorithm Variants

Notice how in the Dual reduction the dependence on the training data only appears in the inner product term. We can generalize from the simple inner product term in a number of ways, and in doing so arrive at the SVM Kernel generalization. The choice of kernel eliminates the need for refining a choice on feature vector mappings beyond a certain point, such as requiring some consistent normalization on feature vectors, for example which is made consistent with choice of kernel (e.g., one could take a discrete probability distribution as feature vector, with its L_1-norm, and its pairing with the entropic kernel). The SVM kernels used in the analysis are referred to as 'Occam's Razor', or 'Razor' kernels. All of the Razor kernels examined perform strongly on channel current data, often outperforming the Gaussian Kernel. The kernels fall into two classes: regularized distance (squared) kernels; and regularized information divergence kernels (both distances squared and information divergences satisfy the triangle inequality – the triangle inequality appears to be fundamental in information modeling in subtle ways). The first set of kernels strongly models data with classic, geometric, attributes or interpretation. The second set of kernels is constrained to operate on $(\boldsymbol{R}^{+})^{N}$, the feature space of positive, non-zero, real-valued feature vector components. The space of the latter kernels is often also restricted to feature vectors obeying an L_1-norm = 1 constraint, i.e., the space of discrete probability vectors. In all of the data-runs with the probability feature vector channel current data considered in (38), the two best-performing kernels are the entropic and the indicator 'Adbsdiff' (or 'Variational') kernels, with the Gaussian trailing in performance in general (but still outperforming other methods such as polynomial and dot product). The L_1-norm channel current feature vector components appear to encapsulate a key constraint of a discrete probability vector via its domain selection and its associated optimal kernel sets.

If the distance term in the Gaussian is denoted $d_G = |\mathbf{x}_i - \mathbf{x}_j| = \sqrt{(\sum_k (x_j^k - x_i^k)^2)}$, the Gaussian Kernel can be written as $K_G(\mathbf{x}_i,\mathbf{x}_j) = \exp(-(d_G)^2/2\sigma^2)$. In general, exponential regularization of a metric on the feature vectors, as in the Gaussian, will provide a Kernel satisfying Mercer's conditions. Since the "kernels" considered in what follows are an extension from those justified by the geometric heuristic to those justified by an information-theoretic heuristic (the final arbiter of performance being empirical results), ***the key property from the above, in obtaining alternate kernels, will be the exponential regularization***.

It is found that ***the other key property is a stability property*** that ties together the best performing kernels from the various cases. For the Gaussian kernel, the stability property is exhibited when the log Kernel variation on feature vector components is calculated:

$$\partial \ln (K_G(\mathbf{x}_i , \mathbf{x}_j)) / \partial x_i^k = (x_j^k - x_i^k)/\sigma^2,$$

where "x_i^k" is the k^{th} component of the i^{th} feature vector and "stability" is indicated by the sign of the difference term $(x_j^k - x_i^k)$, e.g., for

$$K_G(\vec{y},\vec{z}) = \exp(-\| \vec{y} - \vec{z} \|^2 / 2\sigma^2) : \frac{\partial \ln K_G(\vec{y},\vec{z})}{\partial y_K} = (y_K - z_K)/\sigma^2$$

Clearly, the sign is important, as is a notion of difference. Suppose we generalize on this basis to decouple the sign (stability in orientation) convention from the "notion of distance", here providing a new kernel expression, for the "variational kernel" by way of an integration factor:

$$\frac{\partial \ln K_G(\vec{y},\vec{z})}{\partial y_K} = \left(\frac{-1}{2\sigma^2}\right)\left(\frac{\text{sign}(y_k - z_k)}{\sqrt{\sum_k |\, y_k - z_k \,|}}\right)$$

$$K_v(\vec{y},\vec{z}) = \exp\left(-\sqrt{\sum_k |\, y_k - z_k \,|}\,\Big/ 2\sigma^2\right)$$

The subscript "V" in K_V is meant to denote "variational" kernel (sometimes referred to as "indicator" kernel or "Absdiff" kernel). For suitable choice of tuning parameter σ, the variational kernel offers the best performance on the data sets considered. The regularized distance in K_V is the square root of the "Variational" distance: $V(\mathbf{x}_i || \mathbf{x}_j) = \sum_k | x_j^k - x_i^k |$. It is found that the variational kernel is usually the best performing kernel on the L_1 normed data considered in the channel current analysis (L_1 norm: $|x|_1 = \sum_k |\, x_k \,|$, a discrete prob. dist if $x_k > 0$ also). The argument of the exponential in the variational kernel is a distance squared (which satisifies the triangle inequality, etc.), with $K_v = \exp(-d_v^2 / 2\sigma^2)$, thus the variational kernel automatically satisfies Mercer's conditions.

Consider now the case where the notion of difference is not arithmetic but multiplicative, i.e., based on $(1-z_k/y_k)$ rather than (y_k-z_k) (for the Gaussian). In doing so, we must restrict to $y_k \neq 0$ of course. As before, the sign of (y_k-z_k) is information preserved in $(1-z_k/y_k)$, but the latter is not integrable. However, $\ln(y_k / z_k)$ also provides sign info -- positive when $y_k > z_k$, etc., as before, and also includes a ratio. Which to go with? A combination seems best as this is integrable:

$$\frac{\partial \ln K_G(\vec{y},\vec{z})}{\partial y_K} = \left(\frac{-1}{2\sigma^2}\right)\left[\left(1 - \frac{z_k}{y_k}\right) + \ln\left(\frac{y_k}{z_k}\right)\right]$$

$$K_\sigma(\vec{y},\vec{z}) = \exp\left(-[D(y \,||\, z) + D(z \,||\, y)] / 2\sigma^2\right)$$

This is usually a close 2nd to the K_v kernel, sometimes outperforming. This kernel relates feature vectors via relative entropy terms:

$$D(y \,||\, z) = \sum_k y_k \ln\left(\frac{y_k}{z_k}\right)$$

The doubly novel aspect of the entropic kernel is that it would be the very first guess if one wanted to generalize from kernels based on exponentially regularized, square distances, to exponentially regularized, symmetrized, divergences (beginning with the most fundamental, symmetrized "relative entropy" also known as the Kullback-Leibler information divergence) .

A comparison of some of the SVM Kernels of interest is shown in Fig. 2, with "regularized" distances or divergences, where they are regularized if in the form of an exponential with argument the negative of some distance-measure squared ($d^2(x,y)$) or symmetrized divergence measure ($D(x,y)$), the former if using a geometric heuristic for comparison of feature vectors, the

latter if using a information divergence heuristic. Results in Fig. 2 are shown for the Gaussian Kernel: $d^2(x,y)=\Sigma_k(x_k-y_k)^2$; for the Absdiff or Variational Kernel $d^2(x,y)=(\Sigma_k|x_k-y_k|)^{1/2}$; and for the Symmetrized Relative Entropy Kernel $D(x,y)= D(x||y)+D(y||x)$, where $D(x||y)$ is the standard relative entropy.

The SVM algorithm variants that have been explored are minimally detailed here: in the standard Platt SMO algorithm, $\eta=2*K_{12}-K_{11}-K_{22}$, while speedup variations are described to avoid calculation of this value entirely. A middle ground is obtained with the following definition $\eta =2*K_{12}-2$; If ($\eta >=0$) $\{\eta \geq -1;\}$ (labeled WH SMO in Fig. 2, with underflow handling and other details that differ slightly in the WH-SMO implementation as well).

SVM-Internal Multiclass

In the formulation in [24], there are 'k' classes and hence 'k' linear decision functions – a description of that approach is given here, but with the decoupling modifications described in [11].

For a given input 'x', the output vector corresponds to the output from each of these decision functions. The class of the largest element of the output vector gives the class of 'x'. Each decision function is given by: $f_m(x) = w_m.x + b_m$ for all m = (1,2,…,k). If y_i is the class of the input x_i, then for each input data point, the misclassification error is defined as follows: $\max_m\{f_m(x_i) + 1 - \delta_i^m\} - f_{yi}(x_i)$, where δ_i^m is 1 if $m = y_i$ and 0 if $m \neq y_i$. We add the slack variable ζ_i where $\zeta_i \geq 0$ for all i that is proportional to the misclassification error: $\max_m\{f_m(x_i) + 1 - \delta_i^m\} - f_{yi}(x_i) = \zeta_i$, hence $f_{yi}(x_i) - f_m(x_i) + \delta_i^m \geq 1 - \zeta_i$ for all i, m. To minimize this classification error and maximize the distance between the hyper-planes (Structural Risk Minimization) we have the following formulation:

Minimize:	$\sum_i\zeta_i + \beta(1/2)\sum_m w_m^T w_m + (1/2)\sum_m b_m^2$,
	where $\beta > 0$ is defined as a regularization constant.
Constraint:	$w_{yi}.x_i + b_{yi} - w_m.x_i - b_m - 1 + \zeta_i + \delta_i^m \geq 0$ for all i,m

Note: the term **$(1/2)\sum_m b_m^2$** is added for de-coupling, $1/\beta = C$, and $m = y_i$ in the above constraint is consistent with $\zeta_i \geq 0$. The Lagrangian is:

$$L(w,b,\zeta) = \sum_i\zeta_i + \beta(1/2)\sum_m w_m^T w_m + (1/2)\sum_m b_m^2 - \sum_i\sum_m\alpha_i^m(w_{yi}.x_i+b_{yi}-w_m.x_i-b_m-1+\zeta_i+\delta_i^m)$$

Where all α_i^ms are positive Lagrange multipliers. Now taking partial derivatives of the Lagrangian and equating them to zero (Saddle Point solution): $\partial L/\partial\zeta_i = 1 - \sum_m\alpha_i^m = 0$. This implies that $\sum_m\alpha_i^m = 1$ for all i. $\partial L/\partial b_m = b_m + \sum_i\alpha_i^m - \sum_i\delta_i^m = 0$ for all m. Hence $b_m = \sum_i(\delta_i^m - \alpha_i^m)$. Similarly: $\partial L/\partial w_m = \beta w_m + \sum_i \alpha_i^m x_i - \sum_i \delta_i^m x_i = 0$ for all m. Hence $w_m = (1/\beta)[\sum_i(\delta_i^m - \alpha_i^m)x_i]$ Substituting the above equations into the Lagrangian and after simplification reduces into the dual formalism:

Maximize:	$-\frac{1}{2}\sum_{i,j}\sum_m(\delta_i^m - \alpha_i^m)(\delta_j^m - \alpha_j^m)(K_{ij} + \beta) - \beta\sum_{i,m}\delta_i^m\alpha_i^m$
Constraint:	$0 \leq \alpha_i^m$, $\sum_m\alpha_i^m = 1$, i = 1…l; m = 1…k,

where $K_{ij} = x_i.x_j$ is the Kernel generalization, see [11] for details.

SVM Speedup via differentiating BSVs and SVs

If we track the status of support vectors (SV's) according to whether they are Boundary (penalty) or not, and select accordingly, we can get speedup (even in the binary SVM, where choice of C=100 usually good, but C≥10 typically usually okay too). For the multiclass-internal SVM, on the other hand, the speedup with choice of C can be more significant, as shown in [11], where C≥100 typically is needed.

Since the algorithm presented in [24] does not differentiate between SV and BSV, a lot of time is spent in trying to adjust the weights of the BSV i.e. weak data. The weight of a BSV may range from [0, 0.5) in their algorithm. In the modification to the algorithm in [11], as soon as we identify a BSV, its weight is no longer adjusted. Hence faster convergence is achieved without sacrificing accuracy.

Data-rejection heuristics

The SVM Decision Tree shown in Fig. 3 obtained nearly perfect sensitivity and specificity, with a high data rejection rate, and a highly non-uniform class signal-calling throughput. In Fig. 4, the Percentage Data Rejection vs SN+SP curves are shown for test data classification runs with a binary classifier with one molecule (the positive, given by label) versus the rest (the negative). Since the signal calling wasn't passed through a Decision Tree, the way these curves were generated, they don't accurately reflect total throughput, and they don't benefit from the "shielding" shown in the Decision Tree in Fig. 3 prototype. In the SVM Decision Tree implementation described in Fig. 3 [12], this is managed more comprehensively, to arrive at a five-way signal-calling throughput at the furthest node of 16% (in Fig. 3, 9CG and 9AT have to pass to the furthest node to be classified), while the best throughput, for signal calling on the 8GC molecules, is 75%.

The SVM Decision Tree classifier's high, non-uniform, rejection can be managed by generalizing to a collection of Decision Trees (with different species at the furthest node). The problem is that tuning and optimizing a single decision tree is already a large task, even for five species (as in [12]). With a collection of trees, this problem is seemingly compounded, but can actually be lessened in some ways in that now each individual tree need not be so well-tuned/optimized. Although more complicated to implement than an SVM-External method, the SVM-Internal multiclass methods are not similarly fraught with tuning/optimization complications. Fig. 5 shows the Percentage Data Rejection vs SN+SP curves on the same train/test data splits as used for Fig. 4, except now the drop curves are to be understood as *simultaneous* curves (not sequential application of such curves as in Fig. 4). Thus, comparable, or better, performance is obtained with the multiclass-internal approach and with far less effort, since there is no managing and tuning of Decision Trees. Another surprising, and even stronger argument for the SVM-Internal approach to the problem, for many situations, is that a natural drop zone is indicated by the margin.

Suppose we define the criteria for dropping weak data as the margin: For any data point x_i; let $\max_m\{f_m(x_i)\} = f_{yi}$, and Let $f_m = \max_m\{f_m(x_i)\}$ for all $m \neq yi$, then we define the margin as: $(f_{yi} - f_m)$, hence data point x_i is dropped if $(f_{yi} - f_m) \leq$ Confidence Parameter. (For this data set using Gaussian, AbsDiff & Sentropic kernel, a confidence parameter of at least (0.00001)*C was

required to achieve 100% accuracy.) Using the margin drop approach, there is even less tuning, and there is improved throughput (approximately 75% for *all* species).

Single-convergence initialized SVM-clustering

The 'External' SVM Clustering algorithm works by first running a Binary SVM against a data set, with each vector in the set randomly labeled (usually half positives and half negatives), until the SVM converges (Fig. 6). Choice of an appropriate kernel and an acceptable sigma value will affect convergence.

After the *initial* convergence is achieved, the (sensitivity + specificity) will be low. The algorithm now improves this result by iteratively relabeling the worst misclassified vectors, which have confidence factor values beyond some threshold, followed by rerunning the SVM on the newly relabeled data set. This continues until no more progress can be made. Progress is determined by an increasing value of (sensitivity + specificity). With sub-cluster identification upon iterating the overall algorithm on the positive and negative clusters identified (until the clusters are no longer separable into sub-clusters), this method provides a way to cluster data sets without prior knowledge of the data's clustering characteristics, or the number of clusters (by iteration on clusters in the binary SVM or direct with merge of multiclass SVM with label flipping algorithms, described in the Methods and Results, to directly model cluster number without embedded recursion).

The algorithmic variants explored in [11] also explore modifications to tolerate an suitably low number of KKT violators. This is done through running the SVM on the randomly labeled data with different numbers of allowed violators until the number of violators allowed is near the lower bound of violators needed for the SVM to converge on the particular data set. In practice, the initialization step, that obtains the first SVM convergence, typically takes longer than all subsequent partial re-labeling and SVM rerunning steps.

The single-convergence initialized SVM-based clustering algorithm [11] clusters data with no *a priori* knowledge of input classes. The algorithm initializes by first running a binary SVM classifier against a data set with each vector in the set randomly labelled, this is repeated until an *initial convergence* occurs. The convergence sometimes has to be attempted several times (with different randomized initializations) before a SVM solution is obtained. Once an SVM solution is obtained, however, the strengths of the SVM classifier can be used to full advantage. SVMs are ideal in this effort as they not only classify, but offer a confidence parameter with their classification, and can do so in a generalized kernel space. Once a convergent solution is obtained label-flipping (from positive to negative) can be done for low-confidence labels in an iterative process, with SVM re-training after each round of weak-label changes. At each iteration we can potentially have unequal numbers of positives and negatives changing their labels, thus, asymmetrically sized clusters can be realized from a half-positive/half-negative initialization. This iterative process continues until there is no longer a low-confidence classification by the SVM, or until an external cluster validation, such as the sum-of-squared error (SSE) on each cluster, remains relatively unchanged. There are numerous tuning parameters in the SVM-classification process itself, as well as in the SVM-clustering halting specification, and even tuning choices in the SVM chunk-training (that may be necessary for larger data sets). As shown in Fig. 7, SVM-based clustering [11] often outperforms other methods.

Although convergence is always achieved with the single-convergence SVM-clustering method in the label-flippings, after the initial convergence, convergence to a *global* optimum is not guaranteed. Fig. 8 shows the Purity and Entropy (with the RBF kernel) as a function of Number of Iterations, while Fig. 8 (right) shows the SSE as a function of Number of Iterations [37]. The stopping criteria used for the algorithm is based on the unsupervised (external) SSE measure. Comparison to fuzzy c-means and kernel k-means is shown on the same dataset (the solid blue and black lines in Fig. 8 left and center).

The pathology of the single-convergence SVM initialization to get stuck in local minima motivates the introduction of perturbations into the methods, as shown in Fig.s 9 & 10. It is found that the result of the Re-labeler algorithm can be significantly improved by randomly perturbing a weak clustering solution and repeating the SVM-external label-swapping iterations as depicted in Fig. 10. To explore this further, a hybrid SVM-external approach to the above problem is introduced to replace the initial random labeling step with k-means clustering or some other fast clustering algorithm. The initial SVM-external clustering must then be slightly and randomly perturbed to properly initialize the re-labeling step; otherwise the SVM clustering tends to return to the original k-means clustering solution. A complication is the unknown amount of perturbation of the k-means solution that is needed to initialize the SVM-clustering well -- it is generally found that a weak clustering method does best for the initialization (or one weakened by a sufficient amount of perturbation).

Distributed SVM Learning (Chunking Protocols)

Distributed learning on SVMs can be accomplished by breaking the training set into smaller chunks, running separate SVM processes on each of those chunks, and pooling the information that is 'learned', e.g., the support vectors identified as well as nearby (in terms of confidence value) training data vectors and outliers. The reduced pool of data is randomly repartitioned into another round of chunk processing. This is repeated until only a single chunk remains, whose solution is then either the solution sought or close to it (other minor refinements could be sought). There is a fundamental memory limit encountered with larger SVM training sets, such that chunking is needed on training sets even if we aren't interested in the distributed learning speedup (e.g., we need to use a sequential process on a single machine). For this circumstance, for sequential processing of chunks, we take the SV's identified from the prior round of chunk training and merge it with the next chunk to be trained, and iterate. In this way we never have a pure SV training set. If multiple CPU's are available, we can distribute the processing on chunks amongst the machines, and pool their SV's (i.e., pass their SV's to the training pool for the next round). The resulting 'pure SV' training sets, however, are often found to not converge, thus requiring more care to avoid convergence pathologies.

Chunking the SVM learning process becomes a necessity when training (and classifying) large datasets. The number and size of the chunks depends on the size of the dataset to be trained. When training on an individual chunk is complete, the resulting trained feature vectors split into distinct sets (support vectors, polarization set, penalty set, and KKT violator). If the SVM learning is done well, the largest set consists of the support and polarization feature vectors. The polarization set consists of the feature vectors that have been properly classified. These feature vectors pass the KKT relations and have an alpha coefficient equal to zero. The penalty set consists of the feature vectors which pass the KKT relations, and have alpha coefficients equal to C (the max value). The KKT violators make up another set consisting of feature vectors that

violate one of the KKT relations. (The KKT violator set is usually zero at the end of the training process, unless some minimal number of violators is allowed upon learning completion.) These sets give the user different categories of feature vectors that they can pass to the next chunk(s). To keep the SVM converging to a better solution on the next chunk run, however, support vectors (and sometimes some of the polarization set) are passed to the next chunk(s). The optimal pass-percentages of each feature vector set depend on which kernel is used and the dataset.

There are different methods of extracting the feature vectors from the different sets. The specified percentages of feature vectors are pseudo-randomly chosen from each of the sets except for one. *The support feature vectors extraction method differs since it extracts the support vectors that are nearest to the decision hyperplane.* We choose feature vectors whose scores are closer to the hyperplane in order to pass a tighter hyperplane on to the next chunk(s), and manage accumulation of outliers.

The chunk learning topology used in our distributed approach is slightly different from the Binary Tree splitting described in the Cascade SVM presented in [35]. As discussed above, the large dataset is broken into smaller chunks and the SVM is run on each separate chunk. Instead of bringing the results of paired chunks together, all chunk results are brought together and re-chunked as occurred in the first layer. This process occurs until the final chunk is calculated which gives the trained result. At each training stage, the user has the option to tune the percentage of support vectors and non support vectors to pass to the next set of chunks. Additionally, passed support vectors can be chosen to satisfy some max value (approx. C/10 in cases examined) to produce a tighter hyperplane to better distinguish the polarization sets and eliminate outliers. We also incorporate SVR post-processing in some of the dataruns (method below), where SVR runs as part of the core SVM learning task on each chunk. It uses a user-defined alpha cutoff value for further tuning and can significantly reduce the number of support vectors passed to the next set of chunks (with bias towards elimination of outliers and the large non-boundary alphas). These additional steps reduce the size of the chunks, thus making the algorithm run faster without loss of accuracy. The SVR post-processing also appears to offer similar immunity to the convergence pathology (noted in cases involving 100% SV passing on distributed learning topologies).

SVR Method

Support Vector Reduction (SVR) is a process that is run right after the SVM learning step is complete. Instead of going on to testing data against the training results to get accuracy, we further reduce the support vector set. One way to do this is to coerce some alphas to zero which means they would now fall into the polarization set. Converting the smaller alphas to zeros makes the most sense since a larger alpha indicates that the data point is stronger towards its grouping (polarized sign). This is done using a user-defined alpha cut off value. All alpha values that are under the cut off are pushed to zero. It is not entirely trivial since certain mathematical constraints must be met. The constraint that must be met for this method is the linear equality constraint:

$$\sum_{i=1}^{N} y_i \alpha_i = 0$$

Therefore, the alpha values not meeting the cutoff cannot just be forced to zero unless the value is retained somewhere else in the set. This is done by first sorting the alpha values of the support vectors. Then for each alpha that does not meet the cut off value, the small left over value is added to the largest alpha of the same polarity (further biases towards SVR). Since the list is sorted it can loop through and evenly distribute the left over values through the larger alphas starting with the largest. The reduction process can cut the number of support vectors significantly, while not significantly diminishing the accuracy. Other observations have shown that the easier the dataset to classify, the larger the reduction via this process.

Tuning Methods

Tuning is needed to optimize the choice of kernel & kernel-parameter used by the SVM. This is often handled simply by ranging over a collection of roughly 10 kernel types and each at roughly 10 kernel parameter settings (where each is single-parameter kernel), and to do this only on smaller test sets in the training data, where the time complexity of the SVM training is directly tied to the training-kernel computation, which is quadratic in the number of training instances. Although caching can modify the assumptions on time-complexity, there is generally an approximately quadratic time-complexity in the size of the training instances regardless. Chunking must be used to break past this, or more extensive use of GPU capabilities (where there is still the need to do chunking). Chunking algorithms will be shown to be effective, but susceptible to training-failure pathologies if certain safeguards aren't observed, as will be described in the Results.

Once the small test set is done on the initial kernel screening indicated above, a sub-set of kernel will emerge as best, and these are considered again with larger training sets, eventually allowing selection of a good choice of kernel and kernel parameter. More directed tuning paradigms typically involve simulated annealing in this setting (to be shown later). Algorithmic and implementation parameters can also be considered in the tuning, which means we now have a collage of different parameter types in a coupled optimization task. For this type of generalization, genetic algorithms have been applied with amazing success (but not shown in what follows). These more sophisticated tuning methods may not always be necessary in the SVM classification applications, but will allow for successful classifications in some situations where simple methods do not. In the SVM-based clustering methods to be described in what follows, these tuning methods generally play an important role.

Metaheuristics – Parameter Search and Tuning

Our ability to assess a score, or assign a fitness, allows for a collection of metaheuristics that basically reduce to 'look around and take the best way forward' via a series of tweaks. This isn't possible for some problems, however, because the 'looking around' part isn't that informative, e.g., the fitness landscape has sections that are at a fixed level (with noise variations about that level, for example). This is the larger problem of the simple globalization algorithm, via random restart: if the fitness landscape or configuration space is too large random restart won't offer a solution (even if it can) in a reasonable amount of time. This is where more clever metaheuristics must be drawn upon to extend to a global optimization algorithm.

One of the weaknesses of the brute force random restart approach mentioned is that the parameter 'tweak' involved is with a *bounded* perturbative change, which may *already* exclude

the possibility of reaching the solution sought (given the computational resources and a reasonable amount of time). So one generalization is to allow for tweaks that are unbounded, but in some perturbatively stable way, such as with a Boltzmann factor for regularization, and in doing this we arrive at the Simulated Annealing approach:

The global optimization metaheuristics mentioned thus far suggest more sophisticated configuration selection at the component level, on the one hand, and at the population level, on the other hand, especially as we work with the probabilistic simulated annealing approach and the history-based taboo approach (from taboo learning), especially with the component-based versions of the latter. This is because the population and history aspects point to a general metaheuristics that operates on populations of configurations (or populations of 'agents' that interact as intermediaries to determining a configuration selection). The notion of 'history' also must address the conveyance of this information or 'artifact'. In the case of ACO, described in what follows, this will be via stygmergy.

A fixed-size (or size otherwise constrained) population of configurations can have a birth/death cycle or be static. If it has a birth/death cycle, one popular method is the evolutionary computation approach (Darwinian evolution; asexual reproduction):

Metaheuristic: Evolutionary Optimization
Starting population of parent configurations ("parents") undergoes initial selection according to a cut-off (truncation selection) that is chosen. Those surviving produce offspring, typically via a simple configuration tweak mutation, and those child configurations ("children") are then added to the pool of the current population. Repeat.

Depending on algorithm, the parents may be selected against generationally also (such as for salmon). The reproduction step can be done by the population all at once (generationally), as described here, or individually, out-of-phase, as commonly done with the GA's given next.

Metaheuristic: Genetic Algorithm
Starting population of parent configurations ("parents") undergoes initial selection according to cut-off (truncation selection) that is chosen. Those surviving produce offspring, typically via both simple configuration tweak mutation and non-local configuration component-level swapping, and those child configurations ("children") are then added to the pool of the current population. Repeat.

Once we consider that the configurations in a population may interact with one another, we have a situation where different sub-populations may be given (and now not trivially de-couple), i.e., speciation is possible in the evolutionary population. From there whole ecologies of evolutionary complexity can be developed.

With population-based interactions we can go further and have possible direct coordination between agents: sexual reproduction providing cross-over mutation in GA's, as mentioned above, and swarm activity that provides global information to all agents with action defined accordingly to desired local and global swarm behavior, as defined in what follows.

Metaheuristic: Particle swarm optimization (PSO)

Particle swarm optimization (PSO) also takes its cue from Biology, but not from evolutionary model, but from a swarm model. Here the population is static (the other case than the birth/death cycle case), and there is no selection of any kind. Now the configurations in the population are themselves directly tweaked in response to new information obtained. This is a form of directed mutation and is part of a Lamarckian evolutionary paradigm. The configurations are often viewed as describing particles in a space and the configurations undergo directed mutation, 'motion', in the configuration space, with motion towards the best known configuration, where three levels of knowledge are weighed in the balance: (i) the fittest configuration ascertained by a particular during its history; (ii) the fittest configuration ascertained by the informants of a particular particle (often just a randomly chosen set of particles); and (iii) the fittest configuration discovered by any particle.

Results

Distributed SVM Processing (Chunking)

There are a variety of ways to avoid the pure SV training-set pathology. Since we are interested in training set reduction overall, we consider the possibility of simply reducing the SV set. This appears to work in preliminary tests on well-studied datasets of interest (see Table 1), where the SV's nearest to the decision hyperplane (most supporting the hyperplane) are retained. For the channel current data examined in, with 150-component feature vectors, we find that 30% SV passing is optimal on distributed learning topologies. The low SV-passing percentage that is found to work in *distributed* chunking might fundamentally be an issue of outlier control during distributed learning. Further reduction of SV passed is possible with dropping SV's with confidence values at the other extreme, near zero (i.e., those nearest and most strongly supporting the hyperplane). This entails a additional Support Vector reduction (SVR) process that is run right after the SVM learning step is complete, where we further reduce the support vector set according to some confidence cut-off (actually imposed via cut-off on associated Lagrange multiplier in the SVM/SMO implementation). By reducing the number of support vectors propagated into the next round, we further accelerate the chunked processing. In this way, a strongly performing distributed chunk-training process is possible, with speedup by ~10 in the example shown in the table shown in Table 1 (with no significant loss in accuracy). It appears possible to automate the tuning & selection procedures. To achieve this, it is necessary to examine the stability of the algorithmic parameters such as the pass percentages on the different types of learned data (e.g., see Table 1 for pass percentages indicated).

SVM Method	Sensitivity	Specificity	(SN+SP) / 2	Time (ms)
SMO (non-chunked)	0.87	0.84	0.86	47708
Sequential Chunking	0.84	0.86	0.85	27515
Multi-threaded Chunking	0.88	0.78	0.83	7855
SMO (non-chunked) with SV Reduction	0.91	0.81	0.86	43662
Sequential Chunking with SV Reduction	0.90	0.82	0.86	18479
Multi-threaded Chunking with SV Reduction	0.85	0.83	0.84	5232
Multi-threaded Dist. Chunking with SVR	0.85	0.83	0.84	5973

Table 1. Performance comparison table for the different SVM methods. The distributed chunking used three identical networked machines. Dataset = 9GC9CG_9AT9TA (1600 feature vectors). SVM Parameters: Absdiff kernel (a Razor Kernel, with sigma=.5, C = 10, Epsilon = .001, Tolerance = .001. For chunking methods: Pass 90%

of support vectors, Starting chunk size = 400, maxChunks = 2. For SV Reduction methods: Alpha cut off value = 0.15.

To further enhance processing speed, one can not only perform distributed processing as indicated, but can also boost thread-processing speed on a given computer via use of GPU processing. This has already been undertaken,where distributed chunks of SVM training data were processed using a CPU/GPU that, at marginal added cost (a graphics card), provided as much as a 32-fold speedup on the channel current blockade classification. Similar GPU speed enhancements to the other machine learning algorithms are possible as well.

Stabilized, single-convergence intitialized, SVM-External Clustering

The External SVM Clustering data set is chosen to be an equal positives vs negatives sample of 200 8GC blockade signals and 200 9GC blockade signals (see [12] for details about these molecules). Each feature vector is 150 dimensional and normalized to satisfy the L_1 (norm = 1) constraint. Features from the 8 and 9 base-pair blockade signals were extracted using Hidden Markov Models (for details, see [12]). Although convergence was easily achieved with the External SVM Clustering algorithm (see the Methods), convergence to a global optimum was not guaranteed.

In [11], we see that a small value of Kernel-SSE (herein referred to as SSE) is shown to provide us with a reliable cluster validation measure. The Exteranl SVM Clustering (SVM-Relabeler) algorithm does not use an objective function, and the hope is that by running the algorithm in its purest form, the resulting clusters are reliable solutions. However, running this algorithm in this basic fashion does not consistently provide us with a satisfying clustering solution. In fact, the solution space can be divided into three sets: successful, local-optimum, and unsuccessful. Unsuccessful solutions and local optima solutions are undesirable and the objective is to find a method to eliminate their usage by simply re-clustering for objectively improved clustering (via SSE scoring, for example). Since, the solutions in the unsuccessful set are expected to be easily identified in any experiment that calculates the SSE of a randomly labeled data set, they can be simply eliminated by post-processing. In a control experiment we have randomly labeled the dataset 5000 times and calculated the SSE distribution for the experiment. The resulting distribution has a good fit to Johnson's SB distribution and is illustrated in the histogram of Fig. 11. Using a fitted distribution one can calculate the p-value of a given SSE. For a SSE threshold of 170.5 (accidentally very unlikely) we can directly eliminate the unsuccessful set.

To substantially reduce the local optimum solutions, however, thresholding does not scale well. One solution is to use a simple hill climbing algorithm which is to run the algorithm for a sufficiently long number of iterations to find the solution with the lowest SSE value. To do this, the clustering algorithm is run repeatedly and randomly initialized every time. A solution is accepted as the best solution if it has a lower SSE than the previously recorded value. This can be a very slow learning process, and is a familiar scenario in statistical learning, and one of the popular solutions in those situations works well here as well – simulated annealing.

It is observed that random perturbation by flipping each label at some probability, p_{pert}, is often sufficient to switch to another subspace where a better solution could be found. (Note that $p_{pert} = 0.50$ has the effect of random reinitialization and $p_{pert} = 1$ flips the entire labels.) The hope is that perturbation with $p_{pert} \leq 0.50$ results in a faster convergence. Reliability can be achieved by

searching through the solution space. To do this efficiently, Monte Carlo Methods could be used by taking advantage of perturbation to evaluate the neighboring configuration. The procedure described next uses a modified version of Simulated Annealing to achieve this desired reliability.

As shown in Fig. 12 left, top panel, constant perturbation with $p_{pert} = 0.10$ results in a local-optimum solution that could be otherwise avoided by using a perturbation function depending on the number of iterations of unchanged SSE (Fig. 12 right, top panel). These results were produced using an exponential cooling function, $T_{k+1} = \beta^k T_k$, with $\beta = 0.96$ and $T_0 = 10$. The initial temperature, T_0 should be large enough to be comparable with the change of SSE, ΔSSE, and therefore increase the randomness by making the Boltzman factor $e^{-\Delta SSE / T} \approx e^0$, while β (< 1) should be large enough to speed up the cooling effect.

In the effort shown in Fig.s 9 & 10 (see Methods), it was found that random perturbation and hybridized methods (with more traditional clustering methods) could help stabilize the clustering method, but often at significant cost to its performance edge over other clustering methods (apparently due to getting stuck in local minima traps to which the other parametric clustering methods are susceptible). The 'pure' SVM-external clustering method appears to offer very strong solutions about half the time – which allows for optimization simply by repeated clustering attempts and looking for the most tightly clustered (smallest SSE) solution.This suggested a simulated annealing approach for greater computational efficiency, as shown in Fig. 12 (recent work with Genetic Algorithms, not shown, exhibit even stronger stability). Results of this effort (Fig. 12) significantly improve and stabilize the SVM clustering process.

Given the wide variety of dissimilar tuning parameters in the SVM classification process alone, tests on SVM classification with genetic algorithm (GA) based tuning seems optimal. The very robust and rapid auto-tuning with the GA approach on SVM classification in initial tests strongly suggests that this, or any swarm intelligence search/tuning paradigms, offer important refinement to the SVM-classification efforts and critical refinement to the single-convergence initialization SVM-clustering efforts. For details on the more informed (multiple-conversion initialized) version of the process see the Discussion.

Projection Clustering – clustering in Decision space

Even without modifying the label information (partly informed clustering), there is often strong clustering information in an SVM training solution. One such instance occurs when one set, the positives, are a known signal species (or collection of species). If you have mixture data with known and unknown signal species, and wish to identify (i.e., cluster) the unknown species, then an SVM training attempt with the mixture taken as negatives leads to a cluster identification method via an SVM "projection-score" histogram. (i.e., cluster partitioning in Decision Space). Real channel blockade data has been examined in this way, biotinylated DNA hairpin blockades comprised the positives, and scored as a sharp peak at around 1.0. The mixture signals seen after introduction of streptavidin cluster with scores around 0.5, corresponding to (unbound) biotinylated DNA hairpin signals, and signals that score < –1.0, corresponding to the streptavidin-bound biotinylated DNA hairpins. SVM projection clustering can be a very powerful clustering tool in and of itself, as found in the NTD cheminformatics application.

Discussion

Support Vector Machine methods are described for classification, clustering, as well as aiding with signal analysis and pattern recognition on stochastic sequential data. Analysis tools for stochastic sequential data, Markovian (or causal) data for example, have broad-ranging application in that almost any device producing a sequence of measurements can be made more sensitive, or "smarter," by efficient learning of measured signal/pattern characteristics via HMM/SVM methods. The SVM and HMM/SVM application areas described here include cheminformatics, biophysics, and bioinformatics. The cheminformatics application examples pertain to channel current analysis on the alpha-hemolysin nanopore detector.

Our objective is to establish an automated tuning solution for SVM classification over a variety of novel kernel and algorithmic parameters. In Proof-of-Concept work, this has been done by implementing a genetic algorithm tuning procedure, where SVM performance on training data is used to define a fitness function. In initial efforts with genetic algorithm tuning (in analysis of channel current data), the genetic algorithm tuning results were as good as or better than those obtained by an expert manually, so there is a high degree of confidence that this method will succeed. Alternative, easily distributed, tuning approaches will be considered as needed, and include ACO, and other multi-agent *distributed* intelligence approaches.

Markov-based statistical profiles, in a log likelihood discriminator framework, can be used to create a fixed-length feature vector for Support Vector Machine (SVM) based classification [1]. Part of the idea of the method is that whenever a log likelihood discriminator can be constructed for classification on stochastic sequential data, an alternative discriminator can be constructed by 'lifting' the log likelihood components into a feature vector description for classification by SVM. Thus, the feature vector uses the individual log likelihood components obtained in the standard log likelihood classification effort, the individual-observation log odds ratios, and 'vectorizes' them rather than sums them. The individual-observation log odds ratios are themselves constructed from positionally defined Markov Models (pMM's), so what results is a pMM/SVM sensor method. This method may have utility in a number of areas of stochastic sequential analysis, including splice-site recognition and other types of gene-structure identification, file recovery in computer forensics ('file carving'), and speech recognition.

The Multiple-convergence initialized SVM-clustering approach to unsupervised learning provides another non-parametric means to clustering. In preliminary work, it is found that the SVM-based clustering method also offers prospects for inheriting the very strong performance of standard SVMs from the supervised classification setting. This offers a remarkable prospect for knowledge discovery and enhancing the scope of human cognition – the recognition of patterns and clusters without the limitations imposed by assuming a parametric mode and 'fitting' to it, where resolution of the identified clusters can be at an accuracy comparable to the supervised setting (i.e., where cluster identities are already specified).

The new approach is to first obtain *multiple* SVM convergences at initialization (two might suffice, for example, in many situations) and thereby obtain the confidence magnitudes on data points, and their nearest neighbors (if repeatedly have the same neighbors, have high linkage to them). This is used to inform a label-flipping process to arrive at an improved clustering solution on further iterations and analysis. For example, one approach is to establish a high-linkage high-

confidence label set (labels retained or flipped accordingly) and a low-linkage, low-confidence, label set (some, according to criteria, may be flipped as well, or dropped). The magnitude comparison in the simplest 'multiple' convergence result would involve two convergences, with the difference in confidence value for a particular training instance producing a line segment, and for all the training instances, their two-convergent point-differences would provide a collection of line-segments. The most stable part of the line-segment 'field', that of the high-linkage high-confidence data instances, can then be used, for example, to provide indication of structure to guide tuning efforts and label-flipping criteria.

Conclusions

SVMs are fast, easily trained, discriminators, for which strong discrimination is possible without over-fitting complications. SVMs are firmly grounded as variational-calculus based optimization methods that are constrained to have structural risk minimization (SRM), unlike neural net classifiers, such that they provide noise tolerant solutions for pattern recognition. An SVM determines a hyperplane that optimally separates one class from another, while the structural risk minimization (SRM) criterion manifests as the hyperplane having a thickness, or "margin," that is made as large as possible in the process of seeking a separating hyperplane. The SVM approach thereby encapsulates model fitting and discriminatory information in the choice of kernel in the SVM, and a number of novel kernels have been shown here. SVMs are good at both classifying data and evaluating a confidence in the classifications given, which leaves an opening for use of metaheuristics to bootstrap into a clustering capability, as explored in a number of algorithmic variations in this paper. SVM use in clustering appears to be a very robust platform and, from initial results shown here, promises to be one of the best clustering approaches. In this paper we have shown a number of SVM classification algorithms and new SVM/metaheuristics bootstrap algorithms for clustering.

The SVM implementations described involve SVM algorithmic variants, kernel variants, and chunking variants; SVM classification tuning metaheuristics; and SVM clustering metaheuristics. The SVM training metaheuristics enable use of the SVM's confidence parameter to bootstrap from a strong classification engine to a strong clustering engine via use of label changes, and repeated SVM training processes with the new label information obtained.

SVM Methods and Systems have a broad range of applications: sequential-structure identification, pattern recognition, knowledge discovery, Bioinformatics, Nanopore Detector Cheminformatics, the nanopore transduction detection Nanoscope, and computational engineering with information flows using stochastic sequential analysis tools.

Acknowledgements

The author would like to thank lab technicians Amanda Alba and Eric Morales for help performing the nanopore experiments and University of New Orleans students Anil Yelundur, Ken Armond, and Sepehr Merat for help with the algorithmic implementations and the channel current cheminformatics analysis. The author would like to thank the University of New Orleans, NIH, NSF, NASA, and the Louisiana Board of Regents for research support.

Figures

Fig. 1. Decision boundary (solid line); with margin (region between dotted lines). Instances in the 'wrong' place (double circles) are allowed but incur a penalty.

Fig. 2. Comparative results are shown on performance of Kernels and algorithmic variants. The classification is between two DNA hairpins (in terms of features from the blockade signals they produce when occluding ion flow through a nanometer-scale channel). Implementations: WH SMO (W); Platt SMO (P); Keerthi1 (1); and Keerthi2 (2). Kernels: Absdiff (a); Entropic (e); and Gaussian (g). The best algorithm/kernel on this and other channel blockade data studied has consistently been the WH SMO variant and the Absdiff and Entropic Kernels. Another benefit of the WH SMO variant is its significant speedup over the other methods (about half the time of Platt SMO and one fourth the time of Keerthi 1 or 2).

Fig. 3. Nanopore Detector signal analysis architecture, with use of an SVM Decision Tree for classification.

Fig. 4. The Percentage Data Rejection vs SN+SP curves are shown for test data classification runs with a binary classifier with one molecule (the positive, given by label) versus the rest (the negative). Since the signal calling wasn't passed through a Decision Tree, it doesn't accurately reflect total throughput, and they don't benefit from the "shielding" shown in the Decision Tree in Fig. 3 prototype. The Relative Entropy Kernel is shown because it provided the best results (over Gaussian and Absdiff).

Fig. 5. The Percentage Data Rejection vs SN+SP curves are shown for test data classification runs with a multiclass discriminator. The following criterion is used for dropping weak data: for any data point x_i; if $\max_m\{f_m(x_i)\} \leq$ Confidence Parameter, then the data point x_i is dropped. For this data set using AbsDiff kernel ($\sigma^2 = 0.2$) performed best, and a confidence parameter of 0.8 achieve 100% accuracy.

Fig. 6. SVM-external Re-labeling procedure for clustering.

Fig. 7. Clustering performance comparisons: SVM-external clustering compared with explicit objective function clustering methods. Nanopore detector blockade signal clustering resolution from a study of blockades due to individual molecular capture-events with 9AT and 9CG DNA hairpin molecules [11]. The SVM-external clustering method consistently out-performs the other methods. The optimal drop percentage on weakly classified data differed for the different methods for the scores shown: Our SVM relabel clustering with drop: 14.8%; Kernel K-means with drop: 19.8%; Robust fuzzy with drop: 0% (no benefit); Vapnik's Single-class SVM (internal) clustering: 36.1%.

Fig. 8. SVM-external clustering results. (a) and (b) show the boost in Purity and Entropy as a function of Number of Iterations of the SVM clustering algorithm. (c) shows that SSE, as an unsupervised measure, provides a good indicator in that improvements in SSE correlate strongly with improvements in purity and entropy. The blue and black lines are the result of running fuzzy c-mean and kernel k-mean (respectively) on the same dataset. In clustering experiments in [37], a data set consisting of 8GC and 9GC DNA hairpin data is examined (part of the data sets used in [12]).

Fig. 9: The result of Re-labeler Algorithm with Perturbation. The top plots demonstrate the various Purity and Entropy scores for each perturbed run. The spikes are drops followed by recovery in the validity of the clusters as a result of random perturbation. The bottom plot is a

similar demonstration, by tracking the unsupervised quality of the clusters. Note that after 4 runs of perturbation best solution is recovered.

Fig. 10: (a) and (b) represent the SSE and Purity evaluation of hybrid Re-labeler with Perturbation on the same dataset. Data is initially clustered using k-means to initialize the Re-labeler algorithm. The first segment of the plot (right before the spike at 16) is the result of Re-labeler after 10% perturbation, while the second segment is the result after 30% perturbation. Purity Number of Iterations.

Fig. 11. Nanopore feature vector data (in standard 150 component, L1-norm, format) is randomly labeled 5000 times followed by evaluation of SSE values and production of a histogram of those values as shown. The resulting distribution has a good fit to Johnson's SB distribution.

Fig. 12. (left) Simulated annealing with constant perturbation, (right) Simulated annealing with variable perturbation. As shown in left, top panel, simulated annealing with a 10% initial label-flipping results in a local-optimum solution. In the right panel this is avoided by boosting the perturbation function depending on the number of iterations of unchanged SSE (right, top panel). These results were produced using an exponential cooling function, $T_{k+1} = \beta^k T_k$, with $\beta = 0.96$ and $T_0 = 10$.

References

1. Roux B and Winters-Hilt S. Hybrid SVM/MM Structural Sensors for Stochastic Sequential Data. BMC Bioinf. 9 S9, S12 (2008).

2. Burgess CJC. A tutorial on support vector machines for pattern recognition. Knowledge Discovery and Data Mining, 2(2):121–167, 1998.

3. V.N.Vapnik. The Nature of Statistical Learning Theory. New York: Springer- Verlag, 1995.

4. Girosi F and Poggio T. Regularization algorithms for learning that there are equivalent to multilayer networks. Science, 247:978–982, 1990.

5. V.N.Vapnik. Statistical Learning Theory. New York: Wiley, 1998.

6. Scholkopf B, Williamson RC, Smola AJ. Regularization algorithms for learning that there are equivalent to multilayer networks. Science, 247:978–982, 1990.

7. Doursat R, Geman S, and Bienenstock E. Neural network and bias/variance dilemma. Neural Computation, 4(2):1–58, 1992.

8. Kleinberg J. An impossibility theorem for clustering. Proc. of the 16th conference on Neural Information Processing Systems, (12), 2002.

9. Bishop CM. Neural Networks for Pattern Recognition. Oxford University Press, 1995.

10. Schapire RE and Freund Y. A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci., 55(1):119–139, 1997.

11. Winters-Hilt S, Yelundur A, McChesney C, Landry M: Support Vector Machine Implementations for Classification & Clustering. BMC Bioinf. 7 Suppl 2: S4, 2006.

12. Winters-Hilt S, Vercoutere W, DeGuzman VS, Deamer DW, Akeson M, and Haussler D. Highly Accurate Classification of Watson-Crick Basepairs on Termini of Single DNA Molecules. Biophys. J. 84:967-976. 2003.

13. Crammer K and Singer Y: On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. Journal of Machine Learning Research 2 (2001) 265-292

14. Kapur JN and Kesavan HK. *Entropy optimization principles with applications.* Academic Press; 1992.

15. G.Ratsch K. Tsuda K. Muller, S. Mika and B. Schölkopf, *An introduction to kernel-based learning algorithms*, IEEE Trans. Neural Netw. 12, 2001, no. 2, 181–201.

16. Jaynes E. 1997. *Paradoxes of Probability Theory*. Internet accessible book preprint: http://omega.albany.edu:8008/JaynesBook.html.

17. Kullback S: *Information Theory and Statistics.* Dover; 1968.

18. Amari S; Dualistic Geometry of the Manifold of Higher-Order Neurons. Neural Networks, Vol. 4(4), 1991:443-451.

19. Amari S: Information Geometry of the EM and em Algorithms for Neural Networks. Neural Networks, Vol. 8(9), 1995:1379-1408.

20. Amari S and Nagaoka H: *Methods of Information Geometry.* 2000. Translations of Mathematical Monographs Vol. 191.

21. J. Platt, *Sequential Minimal Optimization: A Fast Algorith for Training Support Vector Machines*, Microsoft Research Tech. Rep. MSR-TR-98-14, 1998.

22. Osuna E; Freund R, and Girosi. F: An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII*. Edited by Principe J, Gile L, Morgan N, and Wilson E. editors. IEEE, New York; 1997: 276-85.

23. Joachims T: Making large-scale SVM learning practical. In *Advances in Kernel Methods -- Support Vector Learning.* Edited by Scholkopf B, Burges CJC, and Smola AJ. MIT Press, Cambridge, USA;. 1998: Ch. 11.

24. Crammer K and Singer Y: On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. Journal of Machine Learning Research 2 (2001) 265-292

25. Hsu CW and Lin CJ: A Comparison of Methods for Multi-class Support Vector Machines. IEEE Transactions on Neural Networks, 13; 2002:415-425

26. Lee Y, Lin Y and Wahba G: Multicategory Support Vector Machines. Technical Report 1043, Department of Statistics, University of Wisconsin, Madison, WI, 2001. http://citeseer.ist.psu.edu/lee01multicategory.html

27. Duda RO, Hart PE and Stork DG, *Pattern classification*, Second Edition, John Wiley and Sons, New York, 2001.

28. Keerthi SS, Shevade SK, Bhattacharyya C and Murthy KRK: Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation,* Vol. 13, 2001:637-649

29. Ben-Hur A, Horn D, Siegelmann HT, Vapnik V: Support Vector Clustering. Journal of Machine Learning Research 2; 2001:125-137.

30. Scholkopf B, Platt JC, Shawe-Taylor, J Smola AJ, Williamson RC: Estimating the Support of a High-Dimensional Distribution. Neural Computation, 13, 1999:1443—1472.

31. Yang J, Estivill-Castro V, Chalup SK: *Support Vector Clustering Through Proximity Graph Modeling.* In *Proceedings, 9th International Conference on Neural Information Processing (ICONIP'02)*, 2002:898-903.

32. Fisher RA. The use of multiple measurements in taxonomic problems. Annals of Eugenics, 7:179–188, 1936.

33. R. Guigo M. Burset, *Evaluation of gene structure prediction programs*, Genomics 3 (1996), no. 34, 353–367.

34. Winters-Hilt, S. and C. Baribault. A Meta-state HMM with application to gene structure identification in eukaryotes. EURASIP Journal of Advances in Signal Processing, Special Issue on Genomic Signal Processing, 2010.

35. Graf HP, Cosatto E, Bottou L, Durdanovic I, and Vapnik VN. Parallel Support Vector Machines: The Cascade SVM, in proceedings NIPS, 2004.

36. Eppstein D and Bern M. Approximation algorithms for geometric problems, pages 296–345. Approximation algorithms for NP-hard problems. PWS Publishing Co, 1996.

37. Winters-Hilt S and Merat S. SVM Clustering. BMC Bioinf. 8 S7, S18 (2007).

FIGURES

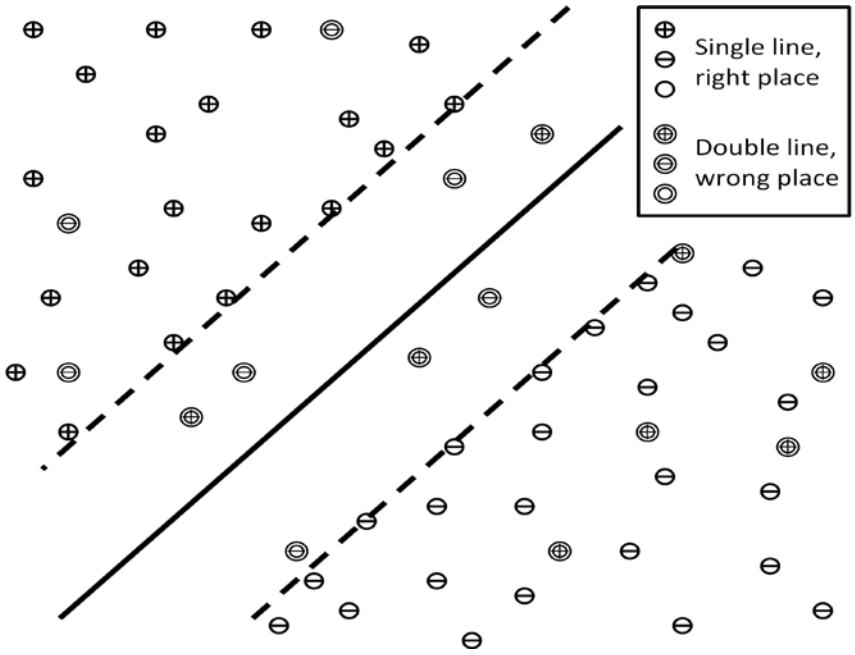


Fig. 1.

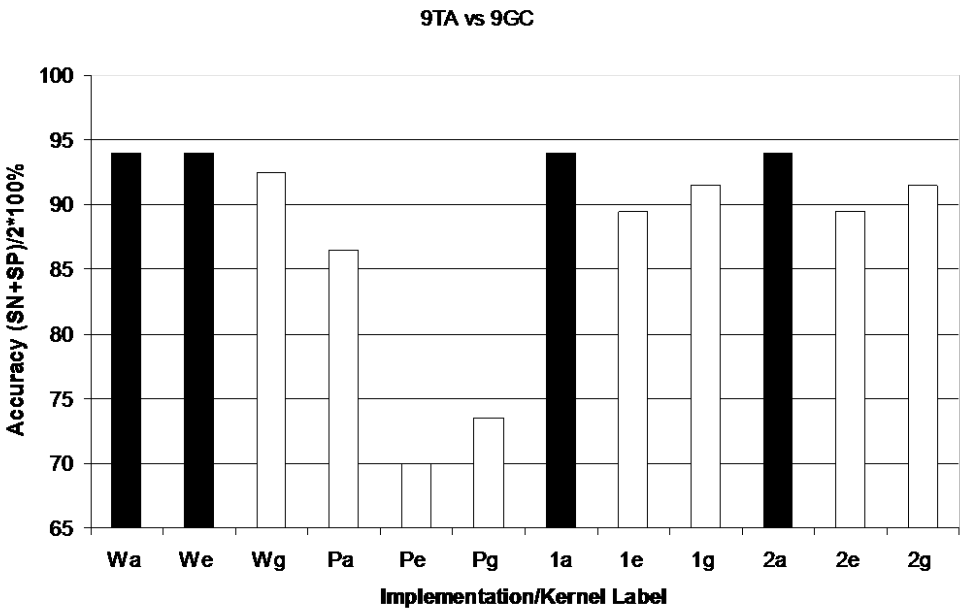

9TA vs 9GC
Accuracy (SN+SP)/2*100%
100
95
90
85
80
75
70
65
Wa
We
Wg
Pa
Pe
Pg
1a
1e
1g
2a
2e
2g
Implementation/Kernel Label

Fig. 2.

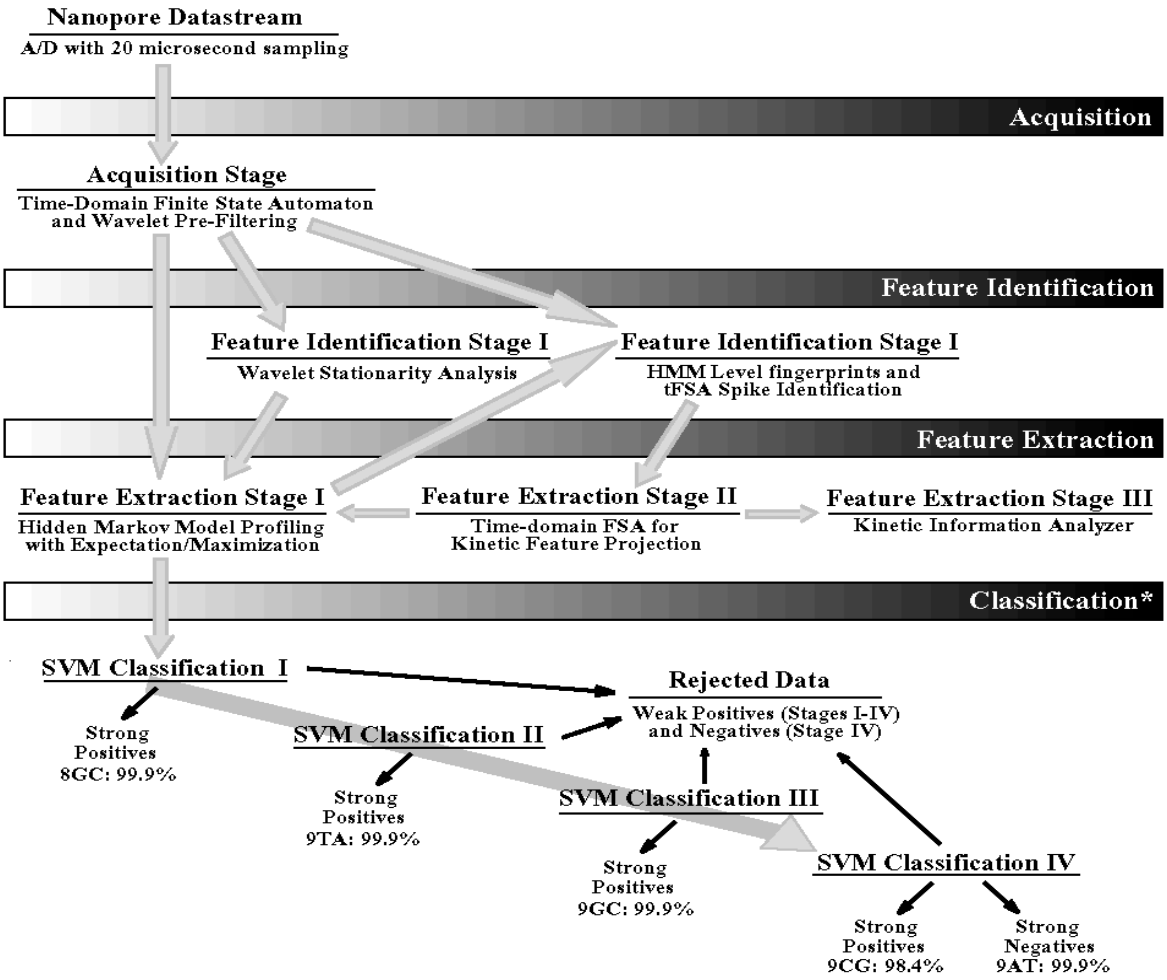

Nanopore Datastream
A/D with 20 microsecond sampling
Acquisition
Acquisition Stage
Time-Domain Finite State Automaton and Wavelet Pre-Filtering
Feature Identification
Feature Identification Stage I
Wavelet Stationarity Analysis
Feature Identification Stage I
HMM Level fingerprints and tFSA Spike Identification
Feature Extraction
Feature Extraction Stage I
Hidden Markov Model Profiling with Expectation/Maximization
Feature Extraction Stage II
Time-domain FSA for Kinetic Feature Projection
Feature Extraction Stage III
Kinetic Information Analyzer
Classification*
SVM Classification I
Strong Positives 8GC: 99.9%
Rejected Data
Weak Positives (Stages I-IV) and Negatives (Stage IV)
SVM Classification II
Strong Positives 9TA: 99.9%
SVM Classification III
Strong Positives 9GC: 99.9%
SVM Classification IV
Strong Positives 9CG: 98.4%
Strong Negatives 9AT: 99.9%

Fig. 3.

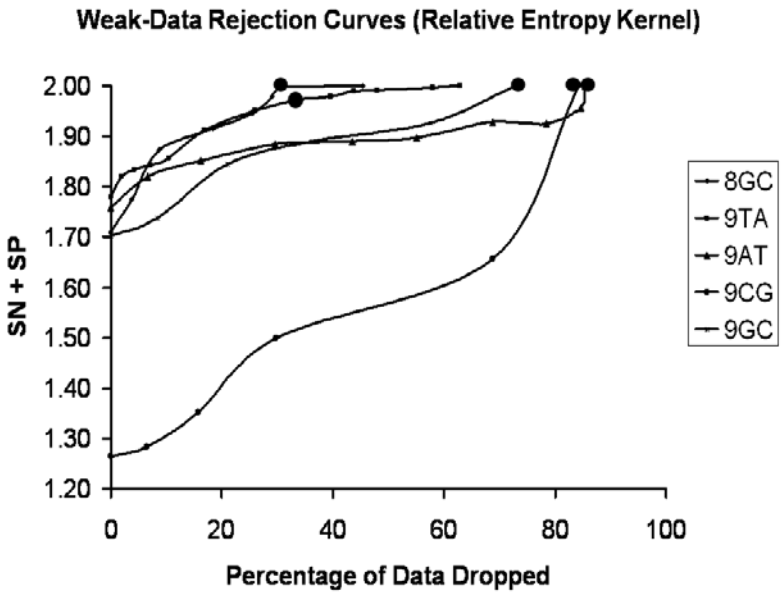

Weak-Data Rejection Curves (Relative Entropy Kernel)
SN + SP
2.00
1.90
1.80
1.70
1.60
1.50
1.40
1.30
1.20
0
20
40
60
80
100
Percentage of Data Dropped
8GC
9TA
9AT
9CG
9GC

Fig. 4.

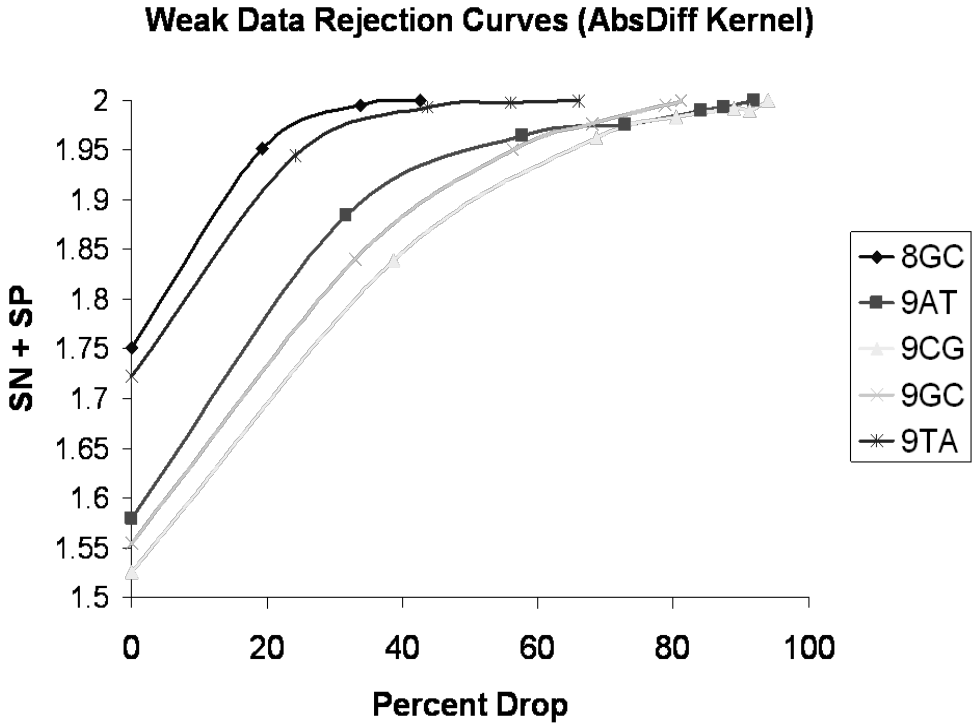

Weak Data Rejection Curves (AbsDiff Kernel)
SN + SP
2
1.95
1.9
1.85
1.8
1.75
1.7
1.65
1.6
1.55
1.5
0
20
40
60
80
100
Percent Drop
8GC
9AT
9CG
9GC
9TA

Fig. 5.

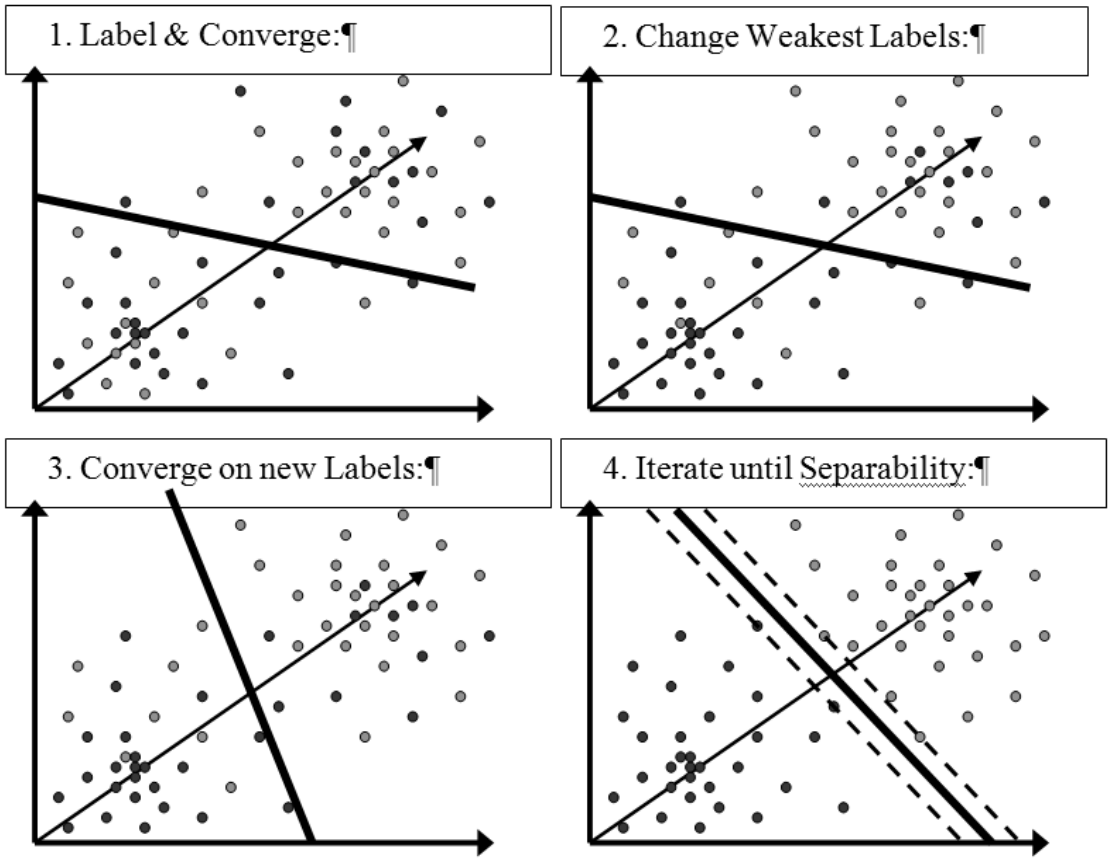

1. Label & Converge:¶
2. Change Weakest Labels:¶
3. Converge on new Labels:¶
4. Iterate until Separability:¶

Fig. 6.

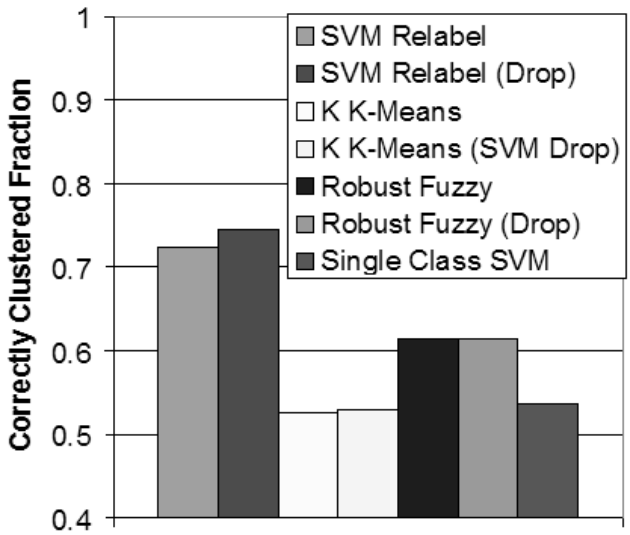

Correctly Clustered Fraction
1
0.9
0.8
0.7
0.6
0.5
0.4
SVM Relabel
SVM Relabel (Drop)
K K-Means
K K-Means (SVM Drop)
Robust Fuzzy
Robust Fuzzy (Drop)
Single Class SVM

Fig. 7.

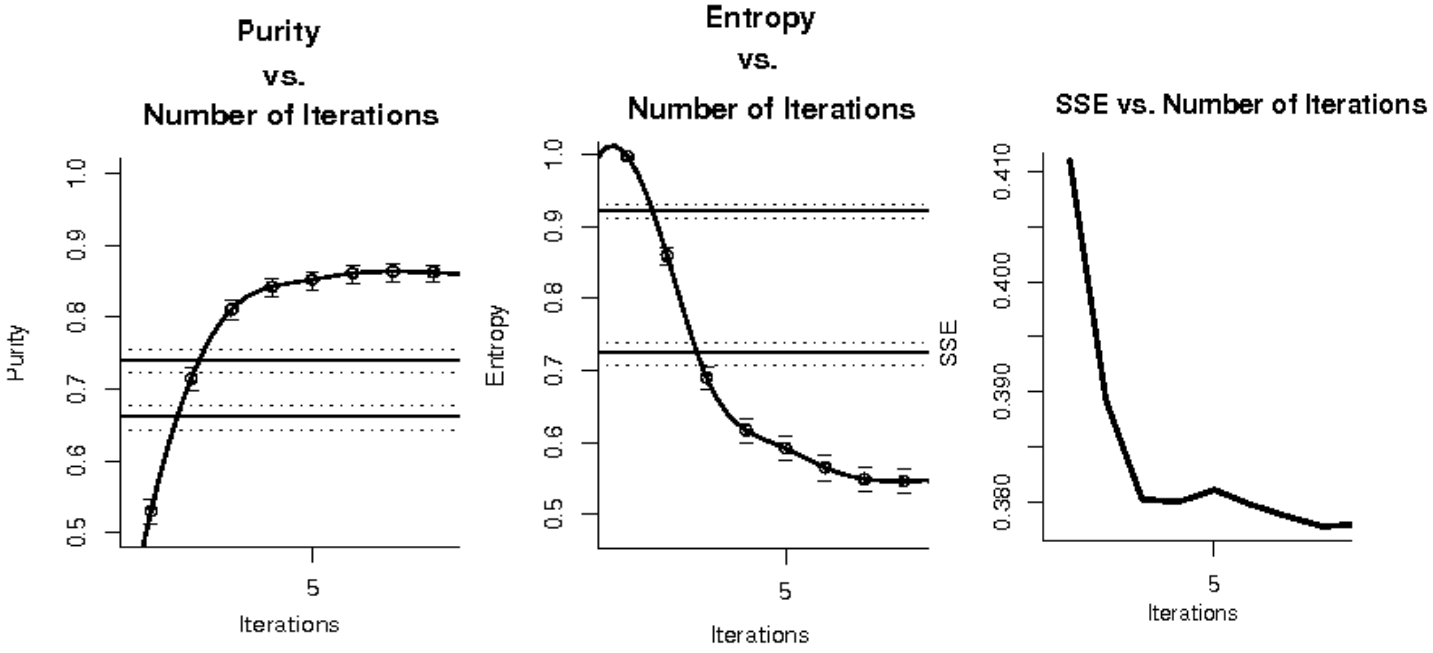

Purity
vs.
Number of Iterations
Purity
Iterations
Entropy
vs.
Number of Iterations
Entropy
Iterations
SSE vs. Number of Iterations
SSE
Iterations

Fig. 8.

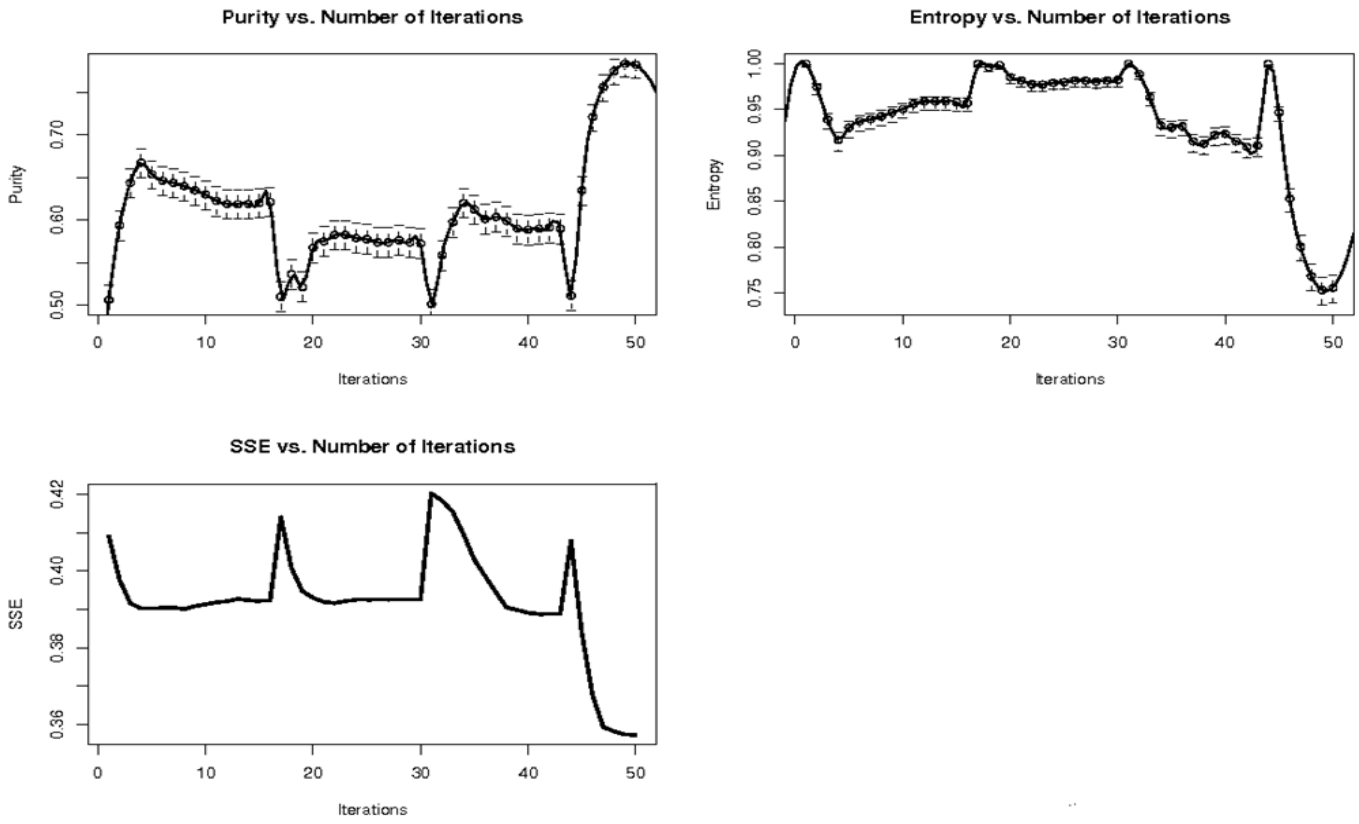
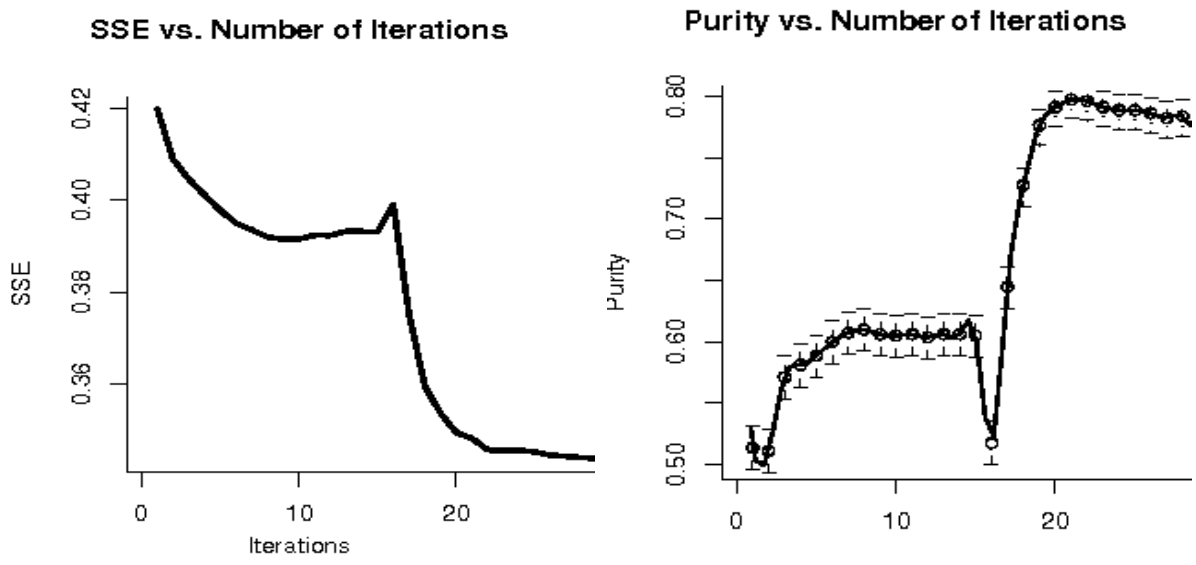

Purity vs. Number of Iterations
Purity
0.50
0.60
0.70
0
10
20
30
40
50
Iterations
Entropy vs. Number of Iterations
Entropy
0.75
0.80
0.85
0.90
0.95
1.00
0
10
20
30
40
50
Iterations
SSE vs. Number of Iterations
SSE
0.36
0.38
0.40
0.42
0
10
20
30
40
50
Iterations

Fig. 9.


SSE vs. Number of Iterations
SSE
0.36
0.38
0.40
0.42
0
10
20
Iterations
Purity vs. Number of Iterations
Purity
0.50
0.60
0.70
0.80
0
10
20

(a) (b)

Fig. 10.

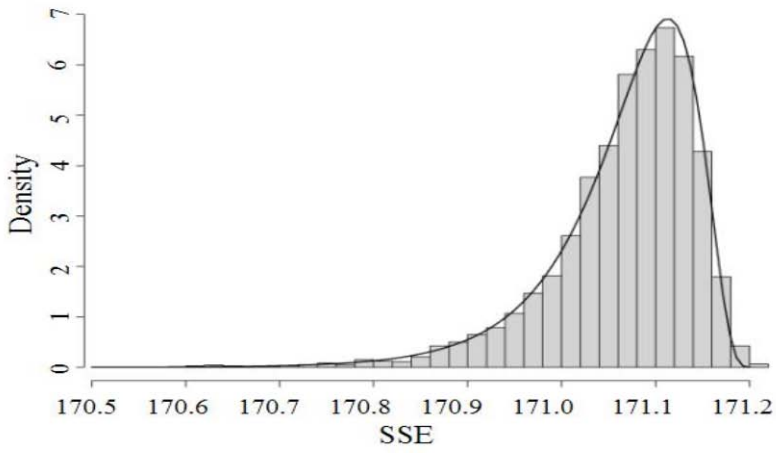
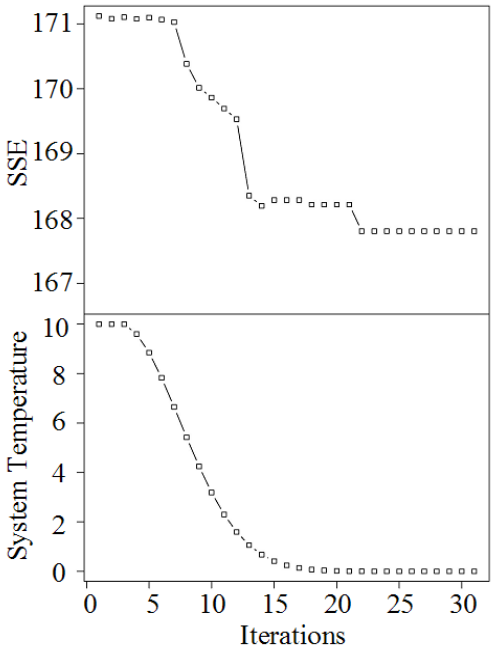

Density
0
1
2
3
4
5
6
7
170.5
170.6
170.7
170.8
170.9
171.0
171.1
171.2
SSE

Fig. 11.


SSE
171
170
169
168
167
System Temperature
10
8
6
4
2
0
0
5
10
15
20
25
30
Iterations

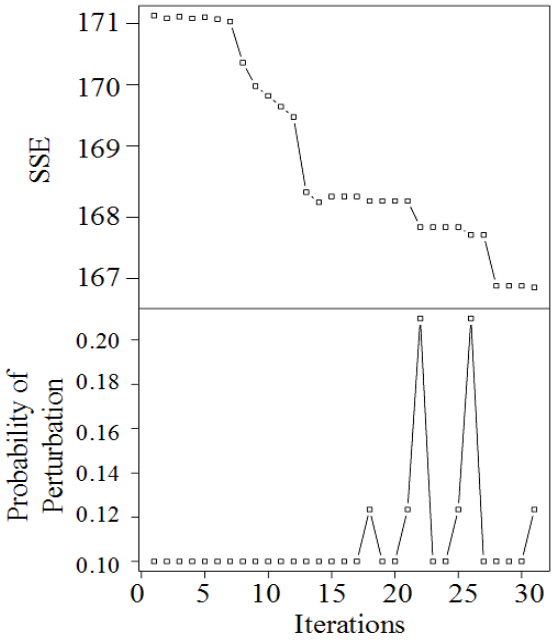

SSE
171
170
169
168
167
Probability of Perturbation
0.20
0.18
0.16
0.14
0.12
0.10
0
5
10
15
20
25
30
Iterations

Fig. 12.