Evolving Kernel Functions with Particle Swarms and Genetic Programming

Michael Schuh and **Rafal Angryk** and **John Sheppard**
Department of Computer Science
Montana State University
Bozeman, MT 59717
{michael.schuh, angryk, john.sheppard}@cs.montana.edu

Abstract

The Support Vector Machine has gained significant popularity over recent years as a kernel-based supervised learning technique. However, choosing the appropriate kernel function and its associated parameters is not a trivial task. The kernel is often chosen from several widely-used and general-purpose functions, and the parameters are then empirically tuned for the best results on a specific data set. This paper explores the use of Particle Swarm Optimization and Genetic Programming as evolutionary approaches to evolve effective kernel functions for a given dataset. Rather than using expert knowledge, we evolve kernel functions without human-guided knowledge or intuition. Our results show consistently better SVM performance with evolved kernels over a variety of traditional kernels on several datasets.

Introducton

The foundation of Support Vector Machines (SVMs) was first developed by Vladimir Vapnik nearly half a century ago. With further advancements and the incorporation of kernel functions, SVMs were popularized in statistical learning theory by (Boser, Guyon, and Vapnik 1992) and (Vapnik 1995). SVMs have been found to be quite effective at linear and non-linear classification, providing yet another alternative for machine learning and data mining tasks.

However, the kernel choice is crucial, as it directly affects the SVM performance. It has been shown that classification accuracy among even the three most widely-used kernels (polynomial, radial-basis, and sigmoidal) can vary significantly based on a given dataset (Howley and Madden 2005). This raises the questions: what kernel should be used when, and with what parameters? Often, the kernel can contain both implicit and explicit domain knowledge, leading to better accuracy, but this requires expert knowledge in the domain of the dataset, as well as the custom creation of a unique kernel function for each task. Moreover, most kernel functions contain several interdependent user-defined parameters that need to be adjusted for optimal performance. It would instead be much more convenient to automatically generate a well-suited kernel function for the given data.

Evolutionary approaches are commonly used on problems which are difficult to approach in a brute force manner. These problems often have loosely formed requirements and focus mostly on the end results through a specific fitness function that determines the effectiveness of a given solution. Evolutionary approaches are based on the biological theories of adaptation and natural selection, and although there is a huge variance in adherence to these principles among algorithms, most involve some sort of fitness-based reproduction between members of a population over a number of generations.

We develop a Particle Swarm Optimization (PSO) and Genetic Programming (GP) framework for evolving kernel functions with the goal of achieving higher supervised classification accuracy than general-purpose kernels. Previous works have shown some positive results using GP and PSO, and we further investigate these approaches through a direct comparative evaluation over a variety of kernels and data. Obviously our framework incurs a higher training cost than just using an existing kernel, but just like normal SVM training and testing, the majority of the time is spent training, whereas testing is a relatively fast operation on the precomputed SVM model. Therefore, depending on the application and domain, the extensive training cost may be worthwhile for a better kernel. Also, manually tuning kernel parameters could still lead to similar training costs over the repeated trail-and-error analysis.

The ability to evolve highly effective kernels could greatly benefit the machine learning and data mining communities. Existing applications can easily replace a kernel after a more effective kernel has been discovered through our approach. The evolutionary framework could also improve the basic usage of SVMs on unfamiliar datasets through assisted exploration of kernel functions and associated parameters, rather than performing countless experiments to analyze these options manually.

The rest of the paper is organized as follows. First the *Background* section explains concepts necessary for understanding the key points of this research and an overview of related work using GP and PSO to create kernels. Then we present and discuss our own *Implementation* details, followed by *Experiments* and a discussion of some specific results, and finally we finish with *Conclusions and Future Work*.

Background

Support Vector Machines

The goal of an SVM is to separate data instances into two classes using examples of each from the training data to define the separating hyperplane. The subset of data instances that actually define the hyperplane are called the "support vectors", and the margin is defined as the distance between the hyperplane and the nearest support vector. By maximizing this separation, it is believed that the SVM better generalizes to unseen data instances, while also mitigating the effects of noisy data or over-training. Error is minimized by maximizing the margin, and the hyperplane is defined as the center line of the separating space, creating equivalent margins for each class. Performance is most commonly evaluated as classification accuracy and/or margin width. Given two SVMs with identical classification accuracy, one would prefer to choose the SVM with a larger margin width, and vice versa. This trade-off is usually incorporated into the training of an SVM.

It is often the case that real world datasets cannot be linearly separated into disjoint classes. However, by transforming the data into a higher dimension (using a kernel function), there more likely exists a linear hyperplane in this new space. When this hyperplane is mapped back to the original data space it becomes a non-linear classification model.

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an effective and flexible technique to explore the search space of a problem. This is partly due to a basic framework that is intuitively simple yet highly extensible. The PSO was first modeled in by (Kennedy and Eberhart 1995) as a simulation of simple social interactions, with relation to flocking birds searching for corn. It contained a group of particles, representing the population, which were randomly placed in the search (or solution) space. Over generations, each particle attempts to move towards an optimum based on personal and neighborhood knowledge of previous best positions. The particle update equations used are given below:

$$v_i = \omega v_i + R(\phi_1) \otimes (p_i - x_i) + R(\phi_2) \otimes (p_g - x_i) \quad (1)$$

$$x_i = x_i + v_i \quad (2)$$

Each particle i of a PSO contains three D-dimensional vectors: the current position x_i, the current velocity v_i, and the previous best position p_i, where D is the dimension of the search space. There is also a best position vector p_g, for each neighborhood which stores the best position of any particle within it. The neighborhood is often defined as the entire population, so p_g can also be referred to as the global best position.

For each generation, the fitness of each particle is calculated, and the personal and neighborhood best positions are updated. Then each particle's velocity and position is updated by the equations 1 and 2, where $R(\phi_1)$ creates a vector of random real numbers between 0 and ϕ_1 that is component-wise multiplied with the difference vector $(p_i - x_i)$ (same for ϕ_2). The variables ϕ_1 and ϕ_2 are the magnitudes for attraction towards the two difference vectors (personal and neighborhood respectively) and are commonly set equal. The inertia weight is defined by ω and acts like a friction variable for the particles trying to move around the search space. Termination criteria for PSO can be a sufficiently good fitness, a set number of generations, or a convergence factor such as a threshold for minimum population change.

Genetic Programming

Genetic Programming was first developed as an evolutionary approach to generating computer programs, initially coined in by (Koza 1992) who used it to evolve LISP programs. In our context, we evolve parse trees containing input data, kernel functions, and mathematical operators, evaluated in post-order fashion. The initial population members (parse trees) are randomly created and genetic information is propagated over generations by standard reproduction techniques.

After parents are selected for reproduction, the crossover function attempts to create diversity and escape local optima by swapping randomly chosen subtrees of each new population member. Unfortunately, a phenomenon known as *code bloat* can cause these trees to grow very large and complex without gaining any significant improvements. These useless sections of trees are termed *introns*, and a common approach to combat code bloat involves assigning a fitness penalty to larger trees. Therefore, if two trees have the same fitness, the smaller tree should be preferred. We also enforce a maximum tree depth to avoid wasted computation time on something which will ultimately (by penalty) have a low fitness. Similar to PSO, termination critiria could be a set number of generations to run, a sufficiently good fitness level, or a convergence factor such as a stagnated population change.

Other Related Works

There has been an abundance of research done with SVMs in the past 10 years. More recently, evolutionary approaches have been explored to aide the training of SVMs. Most of these are split into two general approaches: using genetic programming (GP) to evolve kernel functions, or using particle swarm optimization (PSO) to tune pre-defined kernel parameters. To our knowledge, this is the first work that effectively combines the two methods together and presents a comparative analysis along with an alternative brute-force solution.

The first proposed genetic programming solution to evolving a kernel was by (Howley and Madden 2005). Named the Genetic Kernel (GK) SVM, the algorithm computes the fitness of each population member based on training an SVM with the custom kernel. The fittest members are selected for reproduction, and the new population is again evaluated with SVMs. This process continues iteratively until convergence has been achieved. The combination of GP and the SVM can lead to very high computational overhead, however, results showed that the GK SVM performed better than any traditional kernel across several datasets (although importantly, not always the best for every dataset simultaneously). A similar GP approach was proposed in 2007, and compared against an SVM using grid search to fine tune the parameters for the traditional kernels, results were again promising (Sullivan and Luke 2007). Both papers used a fitness based strictly on classification accuracy. The work of (Diosan, Rogozan, and Pecuchet 2007) also explored a hybrid GP SVM, but used basic normalization functions (Euclidean, Gaussian, and Scalar Product) instead of the traditional kernels, as well as "element-wise operators" which transformed the data while maintaining the same dimension. Although these functions differ slightly, the concepts are the same. They also directly addressed the issue of tree bloat and took corrective actions against it, resulting in evolved kernels with still rather simple, human-understandable structures.

Other works (Feres de Souza et al. 2006) and (Friedrichs and Igel 2005) have applied PSOs to optimize the parameters of traditional kernel functions. In 2009, (Lu, Chen, and Huo 2009) proposed an interesting application of PSOs to not only optimize the kernel parameters, but also the composition of the kernel function itself. The kernel was explicitly defined as: $K_1 \circ K_2 \circ K_3$, where K_{1-3} are general-purpose kernels (polynomial, radial-basis, and sigmoidal), and each $\circ$ is an operator (addition or multiplication). Therefore, the PSO is trying to optimize the set of parameters for all three kernels, as well as the two operators between the kernels simultaneously. Also, fitness here is based on margin size and not classification accuracy. Results were compared against a genetic algorithm (GA) approach and the standard kernels, and tend to show a more generalized SVM with similar accuracy when using their PSO technique.

Implementation

There are many existing software applications offering SVM classification. One of the most popular is Weka (Hall et al. 2009), an open-source collection of machine learning and data mining algorithms developed by the University of Waikato in New Zealand. We use Weka's default SVM implementation based on John C. Platt's sequential minimal optimization algorithm (SMO) (Hall et al. 2009), and then we modified the open source codebase to dynamically set and evaluate a unique kernel function for each population member during run time.

We developed a comprehensive framework containing PSO, GP, and Grid Search implementations based loosely on the combined concepts presented in (Lu, Chen, and Huo 2009), (Sullivan and Luke 2007), and (Howley and Madden 2005). We also reuse Weka's built in Polynomial and Radial-Basis Function (RBF) kernels and include a wide variety of other kernel functions common in literature and very well presented in (Souza 2010). Refer to Table 1 for a complete list of our stand-alone kernel functions. When used in PSO and GP, each kernel is also given a multiplier parameter m to scale its individual result and evolve an overall weighting strategy that (de-)emphasizes the more (less) effective kernel functions.

The PSO implementation was based on the standard framework described in (Poli, Kennedy, and Blackwell 2007), and the search space consists of all the parameters of the custom kernel. For example, a kernel defined as $K = K_0 + K_1 + K_2 + K_3$ where K_{0-3} are the pre-defined stand-alone kernels, would have a particle vector of eleven dimensions: $m_{K_0}, c_{K_0}, m_{K_1}, a_{K_1}, c_{K_1}, d_{K_1}, m_{K_2}, r_{K_2}, m_{K_3}, a_{K_3}, c_{K_3}$, where the m_{K_i} parameters are the included multipliers. This is similar to the work of (Lu, Chen, and Huo 2009), however, they also add the operators between kernel functions to the parameter vector. Discretization of these additional dimensions should be handled carefully because changing the operators between the kernels could mean completely different optimal parameter settings, significantly negating previous swarm exploration, direction, and momentum. Additionally, since our underlying structure of PSO kernels is also represented as parse trees, we have much greater flexibility of defining the custom kernel structure (and parameters) to be optimized. We can easily explore different combinations of operators and kernels and automaticaly apply the PSO framework to tune any given custom kernel.

Table 1: The list of kernel functions used by our PSO and GP framework, with IDs listed for easier reference.

ID	**Kernel Name**	**Equation** $K(x_i, x_j) =$
0	Linear	$((x_i \cdot x_j) + c)$
1	Polynomial	$(a(x_i \cdot x_j) + c)^d$
2	RBF (Weka)	$\exp\left[-\frac{D(x_i, x_j)}{2r^2}\right]$
3	Sigmoid	$\tanh(a(x_i \cdot x_j) + c)$
4	RBF (Alternative)	$\exp\left[-g * \|\|x_i - x_j\|\|^2\right]$
5	Rational Quadratic	$1 - \frac{\|\|x_i - x_j\|\|^2}{\|\|x_i - x_j\|\|^2 + c}$
6	MultiQuadric	$\sqrt{\|\|x_i - x_j\|\|^2 + c^2}$
7	Inverse Multiquadric	$\frac{1}{\sqrt{\|\|x_i - x_j\|\|^2 + c^2}}$
8	Log	$-\log(\|\|x_i - x_j\|\|^d + 1)$
9	Cauchy	$\frac{1}{1 + \frac{\|\|x_i - x_j\|\|^2}{s}}$
10	Histogram Intersection	$\sum_{m=1}^{n} \min(\|x_{i,m}\|^a, \|x_{j,m}\|^b)$

The GP implementation extends naturally from the PSO enhancements, where now each population member is itself a parse tree representing the unique structure and parameters of a custom kernel function. Each internal node is an operator (addition or multiplication) and each leaf contains one of the defined kernel functions and its associated parameters. This is much broader than (Howley and Madden 2005), who use a combination of vector and scalar operators and explicit dot-product evaluation of a symmetric parse tree. Over each generation, each population member's kernel function is trained and tested on the dataset, and awarded a fitness result equal to the tested classification accuracy. To combat code bloat, an empirically-derived penalty of 2% was subtracted from the fitness for every additional level the parse tree contained beyond a given threshold.

Tournament-based selection (with replacement) is performed with a tournament size of two, resulting in a group of parents which when paired together, produce two new children. Before the children are added to the next generation's population, with a specified probability of 80%, crossover will swap a randomly picked subtree between each child. If a swap results in a tree being deeper than the allowed max depth, the first kernel found in the over-sized subtree, through a depth-first search, is moved up to the lowest allowed leaf position. Similar to original GP implementations in literature, we do not use mutation, unlike the 20% probability used by (Howley and Madden 2005). Population members' kernel trees are randomly created upon population initialization, where each leaf node is a randomly selected kernel and then based on the kernel selected, the appropriate parameters are also randomly generated.

Experiments

We perform all experiments on three datasets; two from the popular University of California Irvine (UCI) machine learning repository (Frank and Asuncion 2010), and one real world dataset of NASA solar images created from previous work (Schuh et al. 2012). In our own dataset (referred to as *Dataset 3*), each data instance contains a set of ten numerical features (dimensions) representing an image segment and a class label indicating whether or not the image segment contains a solar filament (binary class). The dataset contains many thousands of instances, so we limit its size to a manageable 1342 total instances – with balanced classes. There are many details about the creation of this data set outside of the scope of this paper, so we refer the reader to (Schuh et al. 2012) for a comprehensive explanation. We chose two UCI datasets that were also used by closely related works. *Dataset 1* is the Pima Indians Diabetes dataset (9 dimensions, 768 instances, 2 classes) used by (Lu, Chen, and Huo 2009), and *Dataset 2* is the Wisconsin Breast Cancer Diagnostic dataset (10 dimensions, 699 instances, 2 classes) used by (Howley and Madden 2005).

Table 2: The mean accuracy (and one-sided 95% CI width) of grid search results for all stand-alone kernel functions.

ID	Dataset 1	Dataset 2	Dataset 3
0	76.97 (5.10)	96.90 (1.69)	73.18 (3.89)
1	77.26 (5.26)	97.37 (1.80)	78.56 (3.39)
2	65.78 (4.04)	83.33 (7.39)	70.13 (5.53)
3	69.95 (6.56)	95.27 (2.22)	54.08 (6.22)
4	65.78 (4.04)	83.41 (7.37)	70.07 (5.65)
5	71.13 (8.16)	96.75 (1.89)	68.40 (3.75)
6	65.78 (4.04)	67.14 (5.25)	48.38 (2.09)
7	65.78 (4.04)	67.14 (5.25)	48.38 (2.09)
8	77.10 (6.01)	97.08 (1.75)	75.55 (3.58)
9	65.78 (4.04)	95.74 (2.55)	55.44 (14.76)
10	76.54 (5.79)	97.01 (1.58)	74.76 (3.93)

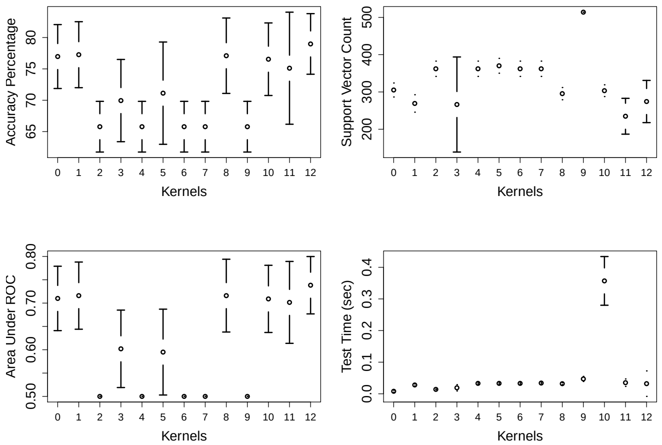

Figure 1: The results (mean values with 95% CI) of the best kernels on Dataset 1, including PSO (11) and GP (12).

Each dataset was split into train, validation, and test sets which were composed of 60%, 20%, and 20% of the data, respectively. All evaluations during any given experimental run contained the same data instances in each data subset, and unless otherwise noted, results are based on 12 runs for each experiment. The general-purpose kernels do not evolve or evaluate fitnesses and therefore are simply trained and tested (validation set merged with training set). The GP and PSO kernel functions from each population member are trained with the training set, and evolutionary fitness (for each generation) is assessed on the validation set. Finally, the best kernels evolved from GP and PSO are evaluated on the unseen test set as the representative kernel functions of that run.

A standard set of statistics is returned from each training and testing of a new SVM with a specified kernel function. The statistics include: kernel calls during training and testing, kernel cache hits during training, number of support vectors needed to define the hyperplane, classification accuracy (percentage correct), area under Receiver Operating Characteristic (ROC) curve, and elapsed time during training and testing. Many of these statistics are omitted from our current discussion because they not emphasized by our focused goal of classification accuracy, but the benefit of having these additional and alternative measurements of performance and success readily available is important. Also, countless other experiments could be performed, investigated, and fine-tuned, but as an investigatory paper, we emphasize the potential of our evolutionary framework and highlight this with selected experimental results.

We first present results of stand-alone kernels. Using grid search, we split each parameter into eight equally-spaced values between a specified minimum and maximum range. The major downside of this method is that each extra parameter is exponentially more costly, so we hard-coded the polynomial kernel constant ($c = 1$), limiting all kernels to a max of two parameters, equating to 64 unique "steps" of possible parameter value combinations. The best step is found for each kernel, and we present the classification accuracy results in Table 2, with the one-sided 95% confidence interval (CI) width provided in parentheses.

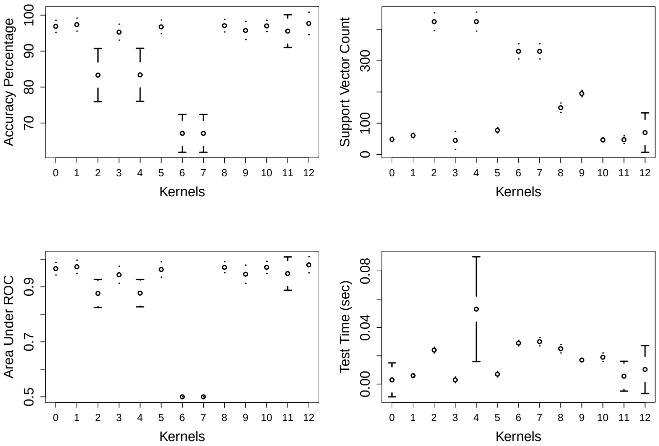

Figure 2: The results (mean values with 95% CI) of the best kernels on Dataset 2, including PSO (11) and GP (12).

We also present these results graphically in Figures 1 and 2, with results included after kernel 10 for the PSO and GP solutions (in that order). Additionally, we include the corresponding results for: support vector count, area under ROC curve, and test time (in seconds). The error bars are the 95% CIs that surround the mean value displayed as an open circle. Certain plots are missing error bars, and this indicates an error range too small to graphically display given the range of the plotted statistic.

Here we can more easily see that some kernels intrinsically perform better than others for the given dataset. We can also see clearly that while kernel 10 has a good classification accuracy, it is drastically slower than all others during testing. For both datasets, the GP and PSO solutions perform equal or better than most other well-suited fine-tuned stand-alone kernels. It is especially important to reiterate that even the stand-alone kernels presented here are the result of extensive (and costly) brute-force searching through vast ranges possible parameter values.

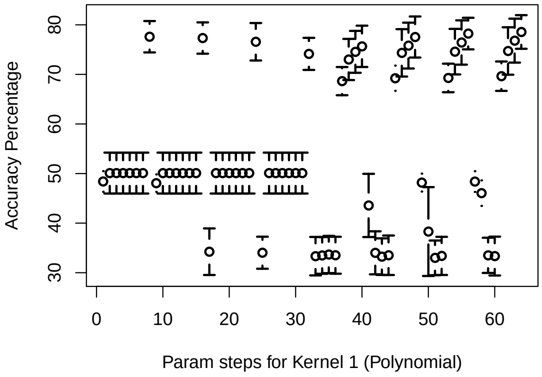

Figure 3: The accuracy results of each grid search step for the polynomial kernel on Dataset 3.

To further illustrate that one cannot blindly apply a kernel function and expect reliable results, we present the accuracy results of each grid search step for the polynomial kernel on Dataset 3 in Figure 3. Effective parameter ranges (and combinations) can be more easily discovered with these simple graphical visualizations, and repetitive patterns over a number of steps indicates certain values of a specific subset of parameters work best together. For example, the best step for the polynomial kernel in Figure 3 is #63, which translates to parameter values of $(c = 1, a = 10, d = 2)$. Interestingly, it can also be seen that when $a > 0$ $(steps > 31)$, the kernel does exceptionally well with $d > 0$, but exceptionally poor with $d < 0$.

Similar to the steps of grid search, we can graphically analyze the steps of PSO or GP runs. For these evolutionary approaches, it might also be worthwhile to know the minimum and maximum fitness over each generation, which are added to the figures as triangle points connected by dotted lines. We report the mean statistical values with 95% confidence intervals (CI) over all members of each generation. However, here we do not aggregate generation statistics over all runs – like we did for grid search steps – because we would lose the details of each uniquely evolved population.

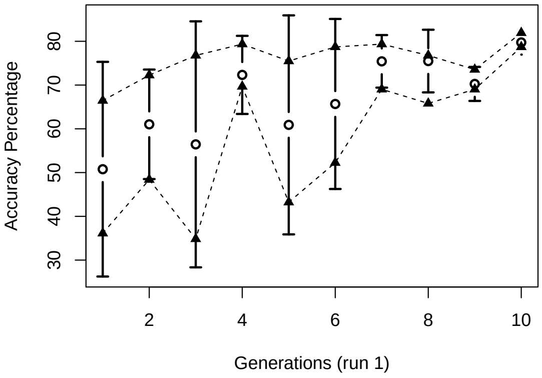

Figure 4: The accuracy results of each generation during a single PSO run on Dataset 1.

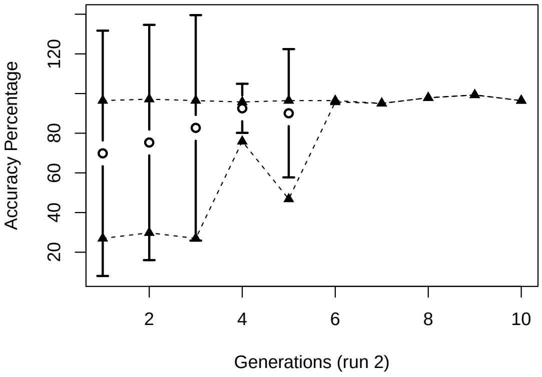

Figure 5: The accuracy results of each generation during a single GP run on Dataset 2.

The accuracy results for a single PSO run on Dataset 2 are displayed in Figure 4. These results look very much like we would expect from a well performing optimization algorithm. Notice the narrowing confidence intervals, implying a converging population. Also notice that all are gradually slowing their increase over each generation, implying a possible optima reached in the solution space. We empirically find that kernel function shows significant parameter improvement after relatively few generations – an ideal finding to maintain lower training time. Similarily, the accuracy results for a single GP run on Dataset 1 are displayed Figure 5. In general, Dataset 1 is a much easier classification problem, and we can see our GP population quickly converging on a near optimal solution and maintaining it.

Conclusions and Future Work

We presented evolutionary approaches to evolve kernel functions that better classify a given data set. Two techniques, Genetic Programming and Particle Swarm Optimization, were applied and compared to a variety of general-purpose kernels on several data sets. Results showed that these methods are feasible, effective, and overall quite promising. We hope this work sparks interest in kernel function evolution and informs the community of readily pursuable work in this area of research.

Our first focus is to perform more comprehensive experiments on our new framework. While the effective ranges for kernel parameters have been looked at through step graphs, many other algorithmic parameters have not been fully explored. This includes parameters of the GP and PSO algorithms as well as the actual SVM implementation, which could also be incorporated into our evolutionary approaches.

An especially intriguing idea is to combine the two concepts into one evolutionary strategy which we hypothesize would speed up the lengthy evolutionary process. Although, this “speed up” comes at the cost of additional complexity, it would be interesting to test if the additional PSO-based parameter tuning would benefit the evolving GP kernel function, perhaps through faster local optima exploration. In other words, the GP would focus on the structure of the kernel, but its fitness would now be determined after a PSO has optimized its parameters – thereby emphasizing the potential of a given structure during reproduction. We would expect the GP to converge in fewer total generations, but at the cost of time taken to compute each generation, which now requires a PSO run for each population member. This raises alternative research opportunities in the parallelization and distribution of these algorithms.

References

Boser, B. E.; Guyon, I. M.; and Vapnik, V. N. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, 144–152. ACM Press.

Diosan, L.; Rogozan, A.; and Pecuchet, J. P. 2007. Evolving kernel functions for svms by genetic programming. In *ICMLA ’07: Proceedings of the Sixth International Conference on Machine Learning and Applications*, 19–24. Washington, DC, USA: IEEE Computer Society.

Feres de Souza, B.; de Carvalho, A. C. P. L. F.; Calvo, R.; and Ishii, R. P. 2006. Multiclass svm model selection using particle swarm optimization. In *HIS ’06: Proceedings of the Sixth International Conference on Hybrid Intelligent Systems*, 31. Washington, DC, USA: IEEE Computer Society.

Frank, A., and Asuncion, A. 2010. UCI machine learning repository.

Friedrichs, F., and Igel, C. 2005. Evolutionary tuning of multiple svm parameters. *Neurocomputing* 64:107 – 117.

Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. 2009. The WEKA data mining software: An update. *SIGKDD*.

Howley, T., and Madden, M. G. 2005. The genetic kernel support vector machine: Description and evaluation. *Artif. Intell. Rev.* 24(3-4):379–395.

Kennedy, J., and Eberhart, R. C. 1995. Particle swarm optimization.proceedings of. In *IEEE International Conference on Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, Vol.IV*, 1942–1948. Press.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, Cambridge, MA.

Lu, M.-Z.; Chen, C.; and Huo, J.-B. 2009. Optimization of combined kernel function for svm by particle swarm optimization. *Machine Learning and Cybernetics, 2009 International Conference on* 2:1160 –1166.

Mierswa, I. 2006. Evolutionary learning with kernels: a generic solution for large margin problems. In *GECCO ’06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 1553–1560. New York, NY, USA: ACM.

Paquet, and Englebrecht. 2003. Training support vector machines with particle swarms. In *Intl. Conf. Neural Networks*, 1598–1603.

Poli, R.; Kennedy, J.; and Blackwell, T. 2007. Particle swarm optimization. *Swarm Intelligence, Issue 1* 1:1942–1948.

Schuh, M.; Banda, J.; Bernasconi, P.; Angryk, R.; and Martens, P. 2012. A comparative evaluation of automated solar filament detection. *Solar Physics - to appear*.

Souza, C. R. 2010. Kernel functions for machine learning applications. http://crsouza.blogspot.com/2010/03/kernel-functions-for-machine-learning.html.

Sullivan, K. M., and Luke, S. 2007. Evolving kernels for support vector machine classification. In *GECCO ’07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, 1702–1707. New York, NY, USA: ACM.

Vapnik, V. N. 1995. *The nature of statistical learning theory*. New York, NY, USA: Springer-Verlag New York, Inc.