University of California, San Diego

An Evaluation of Hyperparameter Tuning Methods in SVM

Huanxi Liu

Advisor:
Professor Ery Arias-Castro

Department of Mathematics

Contents

Chapter 1

Introduction

In many statistical learning models, hyperparameters are important as they can control the overall behavior of a model. A small change in hyperparameters may change drastically in the performance of the model. The traditional way of performing hyperparameter optimization is grid search, which is also known as the parameter sweep. This method performs an exhaustive search through a pre-defined subset of the hyperparameter space of a model. The hyperparameters that provide the best cross-validation score is usually chosen for the model.

The problem of finding optimal hyperparameters can be viewed as an optimization problem, to find the best value of hyperparameters which would optimize the performance of a model. There are other ways of optimizing hyperparameters, such as genetic algorithm (GA) and particle swarm optimization (PSO) for example. Both methods are biology motivated search-based optimization technique that mimic natural selection and animal behaviors respectively. They are known for their generality, versatility and frequently used to find optimal or near-optimal solutions in difficult problems which would otherwise require an extremely long time to solve.

There have been multiple applications of these algorithms in science, engineering and finance. GA can be applied to structure optimization. The main objective in this problem is to minimize the weight of the structure subject to maximum and minimum stress constrains on each member. GA is also used in medical imaging system such as digital subtraction angiographies (DSA) mentioned by the work from Ghaheri et al. [8] and the book by Gao [7] .

Both GA and PSO are iterative algorithms, repeatedly updating a population of individual solutions until the population converges. We could take the advantage of these algorithms by modifying the target function to achieve the goal of optimizing machine learning model's hyperparameters.

There have been works that show PSO and GA's success in improving hyperparameters in multiple machine learning models; for example work by Li [14] and Chen et al. [4] demonstrate PSO's advantage in SVM on forecasting stock index. In a similar manner, this paper will evaluate PSO and GA's effectiveness and efficiency in optimizing hyperparameters and benchmark against grid search methods. In addition, it will study if the size and feature dimension of the dataset will affect the overall performance. Lastly, a combination of genetic algorithm and particle swarm is also proposed and evaluated.

This paper will focus on the two hyperparameters in non-linear support vector machine (SVM) method. SVM models depends on a regularization parameter C and the bandwidth for particular kernel functions. The radial basis function (RBF) was used as the kernel function. The value of C controls mis-classification rate, and the bandwidth for RBF kernel commonly influences the model complexity [15].

This paper will first introduce some backgrounds of SVM along with its foundation, the optimal separating hyperplane. Then this paper will cover the details of hyperparameter tuning methods that includes grid search, genetic algorithm and particle swarm optimization. The paper will conduct experiments on synthetic dataset to the evaluate multiple simulated datasets with different size. Lastly, a combination of grid search and genetic algorithm hyperparameter tuning method will be studied on the real dataset. From the experimental results, we offer general conclusion and suggestions for each hyperparameter optimization method.

Chapter 2

Support Vector Machine

2.1 Separating Hyperplane

One important topic to be discussed prior support vector machine is the separating hyperplane. In classification problems, a model tries to separate two or more classes. There is a separating hyperplane that represents the model's the decision, which is also known as the decision boundary. For now we focus on the classifiers that construct linear decision boundaries to separate the data, that is, classifiers that computes a linear combination of the input features to return the predicted label. A prominent example is the perceptron, which rely on the loss function to minimize the distance between misclassified points and the decision boundary [17]. As a result, they find a separating hyperplane by minimizing the distance of misclassified points to their decision boundary. However, there could be more than one solution to the classification problem and it's hard to tell which one is the best solution.

In the figure below from Elements of Statistical Learning [10], it demonstrates that there could be multiple solutions found by algorithms like perceptron.

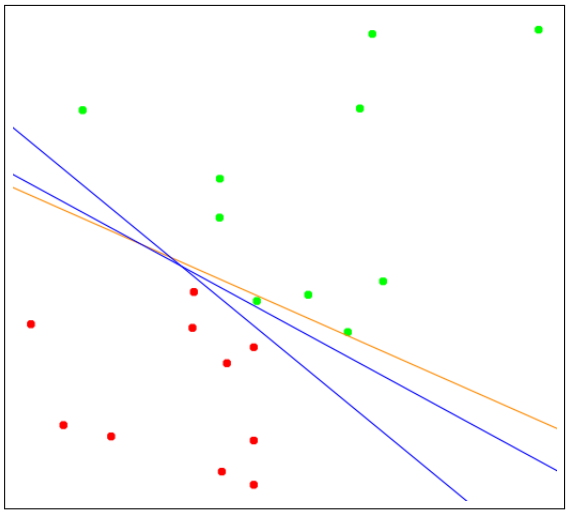

Figure 2.1: A toy example with two classes separable by multiple hyperplanes [10].

There are two classes (colored by blue and green) separated by three hyperplanes. The orange line is the least squares solution, which misclassifies one training point.

The two blue separating hyperplane are found by the same perceptron algorithm with different random initialization value.

In the work by Ripley [16] a number of problems within perceptron algorithm are summarized below:

- In a separable dataset, there are many solutions, the resulting decision boundary depends on the initial value
- In a non-separable dataset, the algorithm takes longer time and may not not converge

This leads us to the problem of locating an optimal separating hyperplane. An optimal separating hyperplane separates two classes while maximizing the distance between the closest point from either class and the decision boundary [6]. This distance is also known as the margin. This method provides a unique solution to the separating hyperplane problem, and the effort of maximizing the margin leads to better classification performance.

With the same data as shown in Figure 2.1, the figure below from the Elements of Statistical Learning shows the optimal separating hyperplane [10].

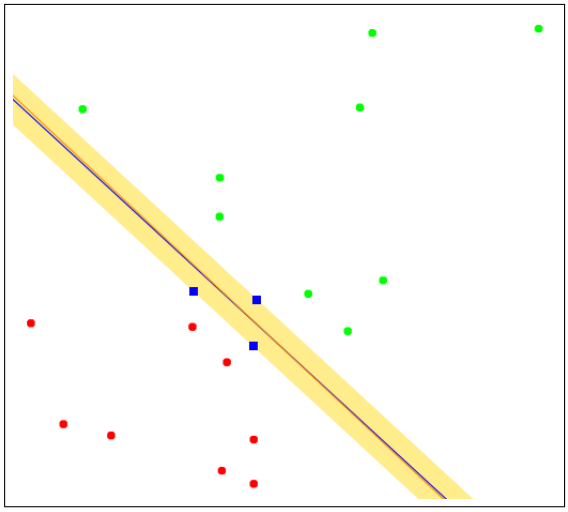

Figure 2.2: Optimal separating hyperplane .

Again the two classes are colored in red and green. The blue line in the middle is an optimal separating hyperplane. (The red line is the decision boundary obtained by logistic regression which is very close to the optimal separating hyperplane). The yellow shaded area represents the vertical distance between points from either class to the decision boundary, which is called the margin. This line is found by maximizing the margin. The three blue colored points are called the support points which were used to produce this decision boundary. It further gives the name of the support vector machine classifiers.

A major difference between the optimal separating hyperplane and other algorithms is that it focuses more on the points that lies around the boundaries between two classes, while algorithms like linear discriminant analysis depends on all of the data, even the ones that arethe far away from the final decision boundary.

2.2 The Support Vector Classifier

The optimal separating hyperplane demonstrated in section 2.1 is in fact a linear support vector machine classifier in a perfect separable dataset. For the rest of this chapter, we describe support vector machine and its classification usage in different scenarios.

Support vector machine (SVM) is a supervised learning model known for strong generalization capabilities for regression and classification. It's developed by Vladimir Vapnik with colleagues [6] at AT&T Bell Laboratories. In binary classification problems, SVM builds model from a set of training sets, each sample is marked one of the two classes. SVM assigns labels to one of the classes predicted, following a non-probabilistic boundary. An SVM classifier tries to maximize the width of the gap between two classes to find the optimal hyperplane that can serve as a boundary. When data are non-linearly separable, SVM adopts a kernel trick to transform the data into higher dimension feature space. Some typical kernel function includes linear kernel, polynomial, radial base function and sigmoid. Each function has parameters associated with the overall model performance.

In a classification problem, consider a training dataset that contain N pairs (x_1, y_1), ... ,(x_N, y_N), where $x_i \in \mathbb{R}^p$ is feature vector with size p and $y_i \in \{-1, 1\}$ is the i-th binary class label. The SVM classifier is defined as a hyperplane given by $f(x) = x^T\beta + \beta_0 = 0$. β is a unit vector such that $\|\beta\| = 1$ as a result of regularization. A SVM makes prediction $P(x)$ by this hyperplane decision boundary:

$$P(x) = sign[f(x)] \tag{2.1}$$

Note that if $y_i f(x_i) > 0$, it indicates a correct classification. We define the margin of a SVM classifier to be M, namely the largest distance between points from the positive and negative class. SVM tries to maximize this margin in finding the most optimal decision boundary, which is the distance from a point x to the hyperplane $f(x) = x^T\beta + \beta_0 = 0$. In a perfect separable case, $y_i f(x_i) > 0\ \forall i$.

$$\max_{\beta,\beta_0,\|\beta\|=1} M \tag{2.2}$$

with subject to $y_i(x_i^T\beta + \beta_0) \geq M$, i = 1,...,N. Note that M=1/ $\|\beta\|$, so equation 2.2 can be re-written as

$$\min_{\beta,\beta_0} \|\beta\| \tag{2.3}$$

with subject to $y_i(x_i^T\beta + \beta_0) \geq 1$, i = 1,...,N.

Suppose that the classes are not separable, we still maximize M; however, we tolerate some points to be on the opposite side of the margin. A slack variable ξ_i is defined to be the distance between the misclassified point to the wrong side of the margin. The margin now is maximized subject to $\sum \xi_i \leq constant$. The problem now becomes

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=0}^{N} \xi_i \tag{2.4}$$

subject to $\xi_i \geq 0$, and $y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \ \forall i$. While the first term controls the margin, the second term associated with the cost C penalizes the number of samples inside the wrong side of the margin. It can be viewed as a level of sensitivity; a high cost would lead to over-fitting with small margin while a small cost would lead to under-fitting with large margin. Hence the value of hyperparameter is crucial.

2.3 Kernel Trick

In case of non-linear separable, a kernel transformation function $K(x, x') = \langle h(x), h(x') \rangle$, is used to enlarge the feature space. We have our prediction in $f(x) = x^T\beta + \beta_0 = 0$. The dimension of the new feature space is allowed to get very large. This is used to help separate the class in higher dimension when it's impossible in the lower dimension. Consider a transformation $h(x)$ in the feature vector, calculating $< h(x), h(x') >$ requires calculating $h(x)$ first, which can be computationally expensive. The kernel trick allows simple calculation that does not need to know the transformation $h(x)$ but only the kernel function, which is fast and helpful in the optimization. Consider two samples x and x', some typical kernel functions are:

- Polynomial kernel: $K(x, x') = (x^T x' + b)^d$ where d is the degree of polynomial function and b is a constant
- Radial basis function kernel (RBF) kernel: $K(x, x') = exp(-\gamma \|x - x'\|)$.
- Sigmoid kernel: $K(x, x') = tanh(\gamma x^T x' + b)$ where b is a constant

Figure 2.3 from R-bloggers [11] demonstrates the SVM boundaries visualized in two dimensions using various kernel functions on the feature space.

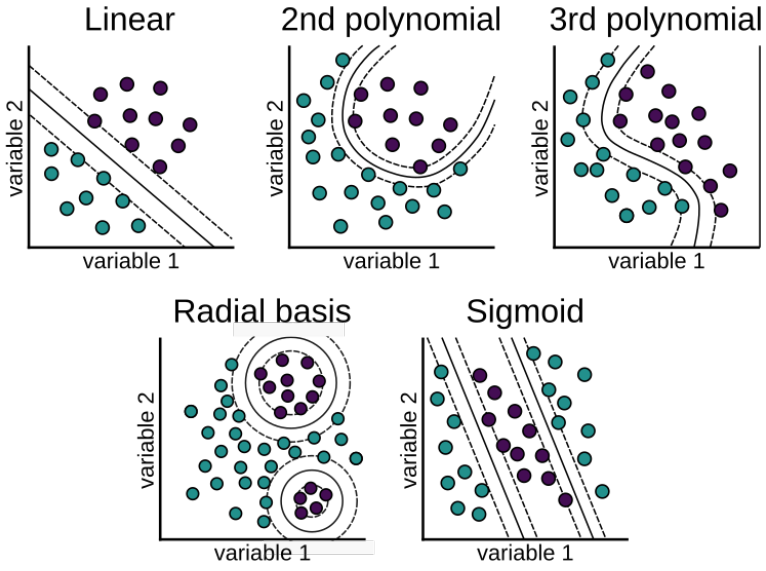

Figure 2.3: Different kernel based SVM boundaries

This paper use RBF kernel for experimental purposes. The γ hyperparameter determines the bandwidth of the model, i.e. how far the influence of a single sample

reaches. Since there is a negative sign in front of γ, low values indicates far influence of one training point and high values indicate close influence of one training point. Because a high γ makes the resulting value of the kernel function is very small. The performance of the SVM model highly depends on the γ parameter; with a low γ, the curve of the decision boundary is relatively flat and thus the decision region is very broad, only includes the support vectors themselves. With a high γ, the curvature of the decision boundary is strong, which generates islands of decision-boundaries around data points that can lead to over-fitting.

Figure 2.4 by Chen [5] demonstrates the effect of γ on the SVM boundaries with a RBF kernel.

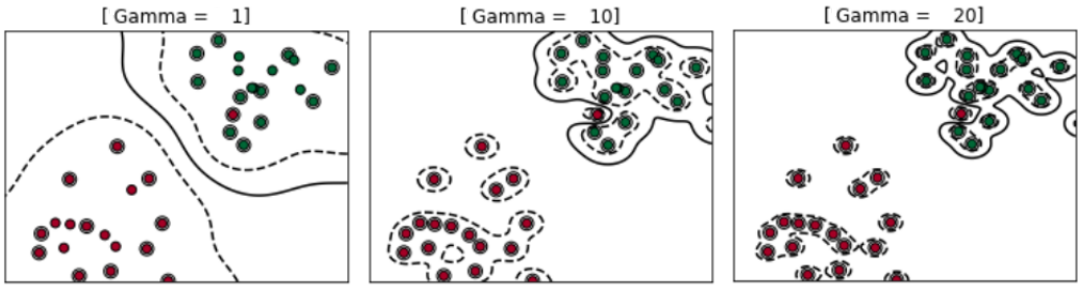

Figure 2.4: RBF SVM with Different Gamma

As we see that the larger the value of gamma, the more complex the classifier becomes. The two hyperparameters C and γ of a RBF kernel is used in evaluating the performance of different hyperparameter tuning methods.

Chapter 3

Tuning Hyperparameters

In this section, we will cover the methods that this paper aims to evaluate and compare. The benchmark test will be made on the grid search method, one of the most straightforward and popular choice in tuning hyperparameters. Then this chapter will describe the essentials of two iterative optimization methods, genetic algorithm and particle swarm optimization, which under specific implementation can be used for tuning hyperparameters.

3.1 Cross Validation

Given a dataset, we could train our machine learning model on this specific dataset. However, achieving high performance on the given dataset may not always be the goal, we wish our model to achieve high performance on the new (unseen) data so we could generalize our machine learning model beyond the given dataset. In fact, when we work too hard on our model to achieve the highest performance on the given dataset, it may pick up the noise and failed to obtain a general prediction rule when used on unseen data [2]. This is often referred as overfitting. Cross-validation is primarily used to estimate the performance of a machine learning model on the unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general [3].

During cross-validation, we purposely hide a portion of the given data to be used to mimic the unseen data and we train on the rest. After the training process, we evaluate our performance on the testing dataset. The procedure is often known as the k-fold cross-validation.

The general procedure of cross-validation is as follows:

Algorithm 1: k-fold cross-validation

Shuffle the dataset randomly;
Split the dataset into k groups;
for *each unique group* **do**
    Take one group as the testing data set
    Take the remaining groups as the training data set
    Fit a model on the training set and evaluate it on the testing set
    Retain the evaluation score and discard the model
Evaluate the model according to the mean of the k evaluation scores

Note that each entry in the dataset is assigned to belong to an individual group and stays for the duration of the procedure. Each entry is guaranteed to be used for testing 1 time and for training k-1 times. In our experiment, a k value of 5 is chosen.

3.2 Grid Search

Grid search is an exhaustive search performed on a the specific value of hyperparameters of a model. Similar to its name, the grid search uses a grid made of different combinations of pre-specified hyperparameter choices. A grid used for SVM with RBF kernel will have the format of the form:

$$\begin{bmatrix} C_1, \gamma_1 & \cdots & C_1, \gamma_M \\ \vdots & \ddots & \vdots \\ C_N, \gamma_1 & \cdots & C_N, \gamma_M \end{bmatrix}$$

assuming there are N choices for C and M choices for γ. Each pair in the grid can be viewed as a potential candidate to the SVM-RBF model and it will be tested out. Grid search can be viewed as a brute-force approach, that is exhaustive but on a limited set of prespecified value. Grid search remains one of the easiest methods to implement when selecting values for hyperparameters. However, trying every candidate out may not be efficient and can be extremely time consuming when searching for an optimal hyperparameter especially when the search space is large. Hence, we introduce the following iterative optimization methods.

3.3 Genetic Algorithm

3.3.1 Motivation

Genetic Algorithms (GAs) were developed by Prof. John Holland and his students and colleagues at the University of Michigan in 1970's [9]. Genetic algorithm is a branch of the field of study evolutionary computation. These algorithms try to solve for the optimal solution of a given function by mimicking the biological processes of reproduction and natural selection. Consider each potential solution as an individual in the entire population. At each time step, the algorithm evaluates each solution's fitness value, according to a predefined fitness function. The individuals with low fitness will be eliminated, as in natural selection where less fitted individuals will go

extinct. In the next time step, new individuals will be populated based on those surviving individuals in a similar manner of genetic breeding and mutation. As this process repeats multiple times until converge, the final surviving individuals represent the best solution it finds.

3.3.2 Implementation

Algorithm 2: Genetic Algorithm

```
Generate initial population;
Compute fitness of each individual;
while The population has not converged do
    Selection;
    Crossover;
    Mutation;
    Compute fitness of each individual;
end
Termination
```

The basic components to genetic algorithms are:

- Initial population: algorithm begins with a set of individuals, where each individual is a value in the feasible set
- Fitness function: determines how fit an individual is according to the value of the function to be optimized
- Selection: determines which individual will survive and pass their genes to the next generation according to their fitness value
- Crossover: mimics biological process to produce the next generation based on surviving individuals
- Mutation: random mutation of individuals in the new generation
- Termination: Algorithm terminates when the population converges (does not produce offspring significantly different than previous generation). Then the algorithm has provided a set of solutions to our optimization problem.

In the setting of tuning SVM's hyperparameters, each individual contains two genes, one represents the cost C and one represents the γ . The fitness function is set to be the 5-fold cross validation error on the training dataset, the lower the fitness function value, the better the individual.

After computing the fitness value (validation error), if the population has not converged, crossover and mutation will begin. Since both parameters take numerical values, they will be turned into binary representation in order to mimic the genetic sequence in process of crossover and mutation. Crossover combines part of parent 1 and part of parent 2's genetic sequence (binary representation of numerical values)

to produce an offspring. During mutation, a number of places on offspring's genetic sequence will be changed with a low random probability to maintain diversity within the population.

When the population has converged after a number of iterations, the individuals within the population will all have similar genes, each of them representing a pair of hyperparameters C and γ. The output of genetic algorithm will be the best set of hyperparameters it finds to be used in this SVM model.

3.4 Particle Swarm Optimization

3.4.1 Motivation

Similar to genetic algorithm, particle swarm optimization (PSO) is a computational method that optimizes a problem (function) by finding optimal hyperparameters iteratively. Instead of eliminating bad solution candidates, PSO tries to improve a candidate solution regarding to a measure of fitness. PSO was first introduced by Kennedy, Eberhart and Shi [13], the original intent was to simulate social behaviors. Kennedy and Eberhart's book on PSO describes many philosophical aspects and swarm intelligence. An survey of PSO applications is made by Poli [18] helped spread the versatility of PSO.

PSO makes few or even no assumptions about the problem (function) and it is able to search large spaces of candidate solutions. However, methods such as PSO do not guarantee that an optimal solution can ever found. An advantage of PSO is that it does not rely on the gradient of the problem being optimized i.e., the function does not need to be differentiable. This characteristic is what makes PSO performs well in complex problems where differentiation is almost impossible such as tuning hyperparameters.

An intuitive explanation of PSO as follows: consider a swarm of birds (solution candidates) looking for food (optimal solution). Neither of them knows where the food is; however, they are able to communicate with each other and know how close they are from the food (by fitness function). In the next iteration, birds will update their location. Birds who are far from the food will move toward those who are close and the above process will repeat until the population has converged.

For each particle, its location represents a solution to the optimization problem. Its velocity will guide it the direction and the magnitude to move in the next iteration. Each particle is able to communicate with other particles and share location, and velocity.

3.4.2 Implementation

Algorithm 3: Particle Swarm Optimization

Generate initial swarm population;
Compute fitness of each individual;
while *The population has not converged* **do**
    Update *pBest* and *gBest*;
    Calculate velocity for each particle;
    Use *pBest* and *gBest* to update each particle's velocity and location;
    Compute fitness of each individual;
end
Termination

The *pBest* is the position that yielded the best fitness value of a single particle; similarly, *gBest* is the position that yielded the best fitness value of the entire swarm population. After the two best values are determined, the position and velocity of the particles can be updated by the following:

$$v_i^{k+1} = wv_i^k + c_1 r_1(pbest_i^k - x_i^k) + c_2 r_2(gbest^k - x_i^k) \tag{3.1}$$

In the equation, w is the inertia weight on the velocity from previous iteration to prevent from particles moving too rapidly. v_i^k is the velocity of the i-th particle at the k-th iteration, and x_i^k is the current position (solution) respectively. c_1 and c_2 are positive constants, along with w that controls the rate of convergence. r_1 and r_2 are two random uniform variables between 0 and 1, contributes to random exploration in the space.

The formula can be viewed in three components, fist part contributes to the inertia from last interation's velocity. The second component drives the particle moving toward its own best location and the last component brings the particle close to the best location that the entire swarm population has discovered.

After calculating the velocity, the location will be updated with the below equation:

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{3.2}$$

This concludes the updating process of the particles. The fitness value will be recomputed based on the new information and the algorithm will repeat until the population has converged. The final position of the particles will be the solution to the optimization problem (function).

Similarly, the fitness function is replaced with the 5-fold cross validation error on the training dataset for tuning hyperparameters purposes.

Chapter 4

Experiments and Results

4.1 The Experimental Setting

The goal is to compare the effectiveness of tuning parameters with grid search, genetic algorithm and particle-swarm optimization. In order to do so, we use random dataset generated by scikit-learn (also known as sklearn) built-in libraries, make_circle and make_classification and we evaluate the prediction accuracy on a SVM model with RBF. With each dataset, the pair of SVM hyperparameter (C, γ) is being optimized using three methods. To access the effectiveness, equal number of evaluations are set for all three methods: Grid Search is a grid made of 10 choices of C and 10 choices of γ equally spaced between 0.0001 and 10, GA is set with 20 iterations and a population size of 5, PSO is set with 20 iterations and 5 particles. All bring to a total of 100 evaluations for each method.

The size of the generated dataset is also varied to study. Size of the dataset is varied between 200 and 1000 with a step size of 25; at each size, dataset is re-generated 5 times.

4.2 Performance Metrics

To quantify the performance, we chose the parameters obtained from the best 5-fold cross-validation score and evaluate model performance on the testing set as well as their execution time. The performance is evaluated as the percent of data not being correctly classified (percent error). For better visualization purpose, all the testing errors were divided by the error by grid search method. Hence, the error grid search method is at 100, meaning a testing error of 100% with respect to grid search's testing error. If PSO achieves 80%, it means that PSO's testing error is 80% of grid search's testing error.

4.3 Results

4.3.1 Sklearn make_circles Dataset

In this experiment, we tested on the dataset generated using the sklearn's built-in package make_circles. It generates a dataset with a feature vector of dimension 2, which makes a large circle containing a smaller circle in 2-dimensional space. An example is shown in figure 4.1 and 4.2

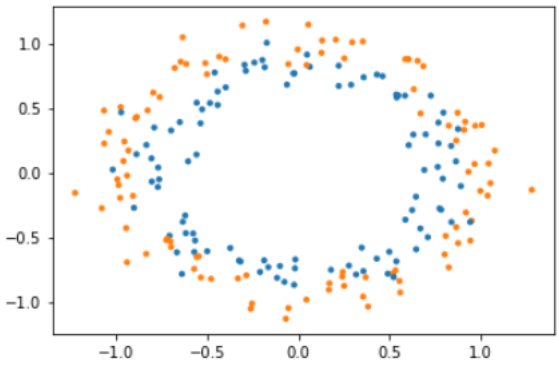

Figure 4.1: Example of make_circles with size 100

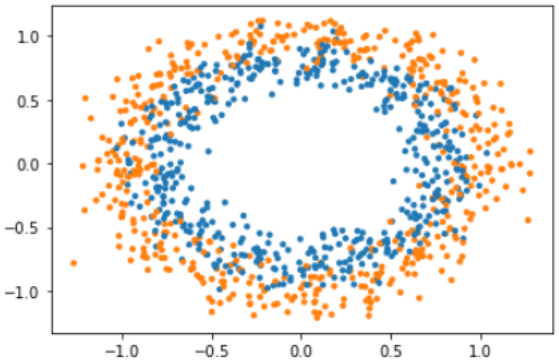

Figure 4.2: Example of make_circles with size 1000

A positive noise level is being used.

To test the effect of data size in the performance of three methods. We experimented with size from 200 to 1000 with a step size of 25. At each size, dataset is generated 5 times.

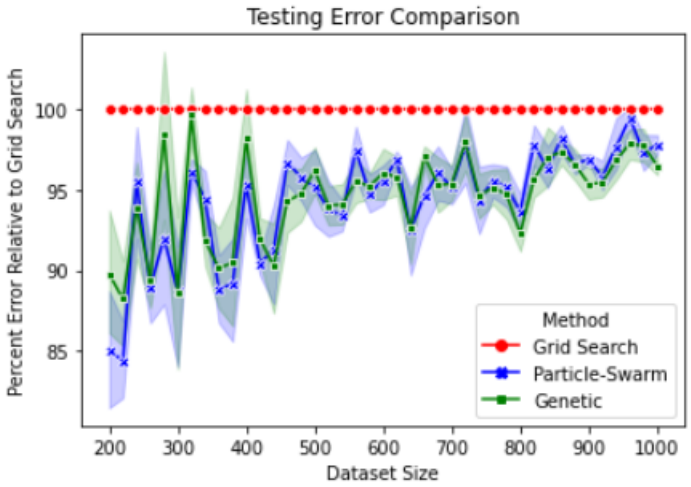

Figure 4.3: Methods Performance in make_circles

The performance comparison of three methods across different dataset size is shown in figure 4.3. Note that the $y - axis$ is the percent error relative to the grid search. Hence the red line is always capped at 100 for grid search. We found that PSO and GA demonstrated better performance in small to middle dataset size. For example, for size equals to 200, PSO and GA achieves about a 10-15 percent

improvement than grid search with same number of evaluations. Three methods achieved similar performance in large dataset. When the dataset is small, there are more freedom in points whereas when the dataset is large, as in figure 4.2, there aren't much improvements from different hyperparameter tuning methods to overcome the noise. It follows the intuition that tuning hyperparameters has minimal effect when facing a large and noisy dataset.

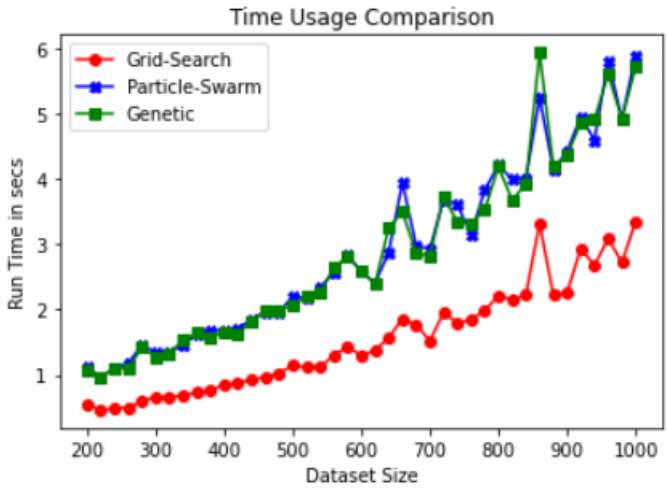

Figure 4.4: Method Time Consumption in make_circles

Figure 4.4 demonstrates the run time for each method. Given that PSO and GA run additional procedures than grid search, the run time usage shows that grid search requires the least amount of run time with same number of evaluations of all methods. The difference is bigger at high dataset size. However, because of the implementation difference, the run time can be further improved. For example, implement back-end in C++ than in Python.

4.3.2 Sklearn make_classification Dataset

We also tested on the dataset generated using the sklearn's built-in package make_classifications. It generates dataset with a feature vector with specified dimension. A sample dataset visualized with two features in 2-dimensional space is shown in figure 4.5

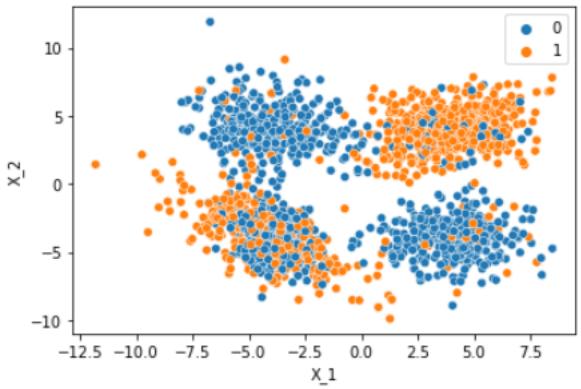

Figure 4.5: Example of make_classification

A positive noise level is also being used to make non-linear separable dataset.

In this experiment, we focused on the effect from different number of features in the performance of three methods. We experimented with a dataset with size 100 and 1000 and number of features range from 3 to 10. At each step, the dataset is re-generated 5 times.

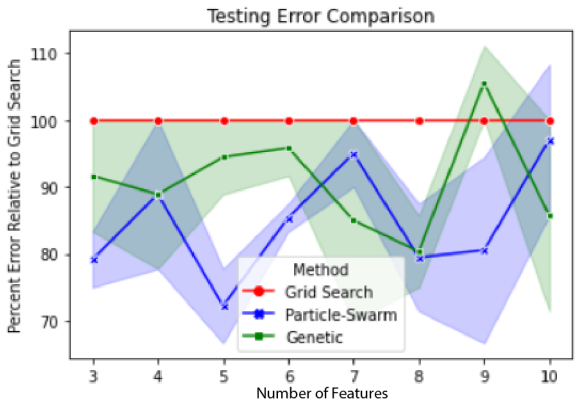

Figure 4.6: make_classifications with size 100

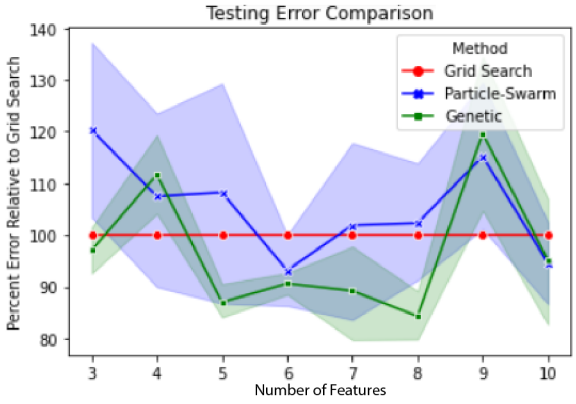

Figure 4.7: make_classifications with size 1000

The performance comparison of three methods is shown in Figure 4.6 and 4.7. The $y - axis$ is again the percent error relative to the grid search. We noticed the consistency that at smaller dataset, PSO and GA demonstrated better performance with some degree of variation. However, when the dataset is large, PSO and GA may not outperform grid search with equal number of evaluations. The performance of PSO and GA can be effected by the large feature space, a higher number of iterations may help achieve a better performance. We noticed an overall higher variance on make_classifications than make_circles, this can contribute to the complete randomness with make_classifications whereas make_circles generates dataset while retaining a circular shape.

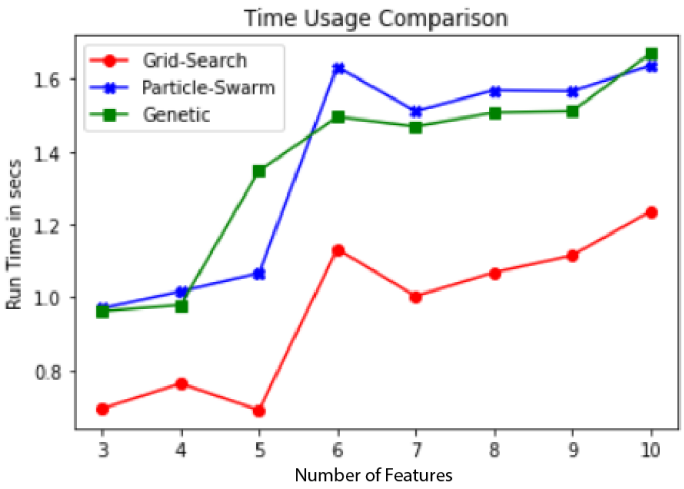

Figure 4.8: Method Time Consumption in make_classifications

Figure 4.8 demonstrates the run time for each method in the smaller dataset (size = 100). The pattern is repeated as in figure 4.4 that the grid search method requires less run time with same number of evaluations for all methods. We also noticed that the run time increases as number of features increases.

Chapter 5

Combination of Grid Search and Genetic Algorithm

5.1 Warm Start Genetic Algorithm

As shown in Chapter 3, we observed the advantage of GA and PSO in finding optimal set of hyperparameters with equal number of evaluations. We could further extend advantage of PSO and GA by initiating a warm start. Note that the boundaries of hyperparameters in SVM are only required to be strictly positive, $C, \gamma > 0$. According to the paper by Hsu et al.[12], exponentially growing sequence of C and γ are recommended, for example from $2^{-11}, 2^{-9}, ..., 2^{13}, 2^{15}$. The vast range of search space results in a huge time and computation consumption for GA and PSO due to their iterative nature. Hence we propose a combination of GS and GA method in tuning SVM's hyperparameters.

We first use grid search to locate a general range of hyperparameter values. Based on grid search's result, we then set the boundaries of C and γ accordingly in the genetic algorithm. The output of the genetic algorithm is the best value of hyperparameters it finds.

5.2 Credit Risk Dataset

Consider this very large and complex dataset, the credit risk dataset. It's real dataset that contains customer's demographic and credit card information such as their age, salary, marital status, credit card limit, credit card category, etc. This dataset is uploaded to Kaggle from LEAPS website [1].

This dataset contains 10127 samples and 19 features, a condensed list of features shown as following:

- Customer_Age: Demographic variable - Customer's age in years
- Gender: Demographic variable - M=Male, F=Female
- Dependent_count: Demographic variable - Number of customer's dependents
- Education_Level: Demographic variable - Educational level of the customer

- ...
- Avg_Utilization_Ratio: Average card utilization ratio
- Attrition_Flag: class label, 0 for existing customer, 1 for attired customer

5.3 Results

With this dataset, a total of three experiments were performed in tuning hyperparameters. The first pair of hyperparameter is done by using grid search only. We used two grids each with 225 evaluations, 15 choices for cost and 15 choices for gamma. The first grid search for range between $2^{-11}, 2^{-9}, ..., 2^{13}, 2^{15}$. As shown in figure 5.1, the cost with value between 2^{1} and 2^{15} along with a gamma value between 2^{-11} and 2^{-9} provides the lowest cv error, the green boxed region. The second grid with equal space is then used again to search for the space described above. There are total of 15x15x2 = 450 evaluations.

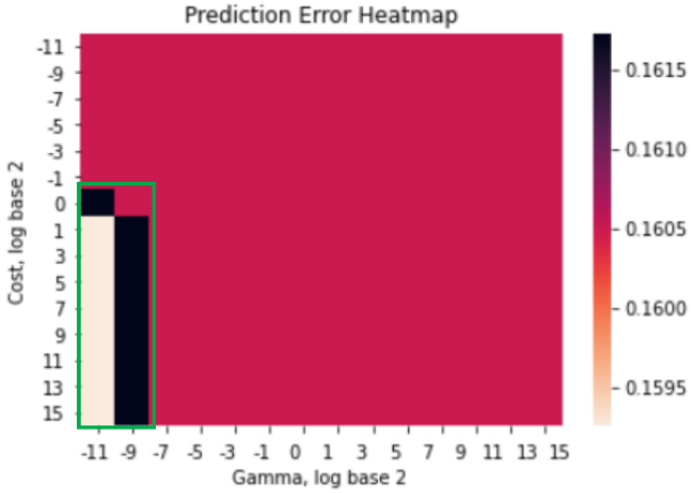

Figure 5.1: Grid Search Heatmap

The second method of tuning is using genetic algorithm alone. The minimum boundary is set to be 2^{-11} and the maximum boundary is 2^{15}. To be comparable with grid search, the population size is set to be 15 with 30 iterations, which brings a total of 450 evaluations.

Lastly, the method of combining grid search and genetic algorithm is used. Grid search is first used to locate a general boundary for hyperparameters between 2^{-11} and 2^{15}. The new range of C and γ is decided according to figure 5.1. This is used to set the boundaries for genetic algorithm and the population size is set to be 15 with 15 iterations. In total there are 450 evaluations.

The results are summarized below:

Method	Time Usage (in seconds)	Testing Error
Grid Search	55.3	15.93%
Genetic Algorithm	88.2	15.93%
GS + GA	67.7	15.88%

Table 5.1: Comparison of methods on credit dataset

We noticed that when using genetic algorithm alone, it didn't obtain a better hyperparameter solution as we seen in the previous experiments. This might caused by the vast space (between 2^{-11} and 2^{15}) to search for while 450 evaluations are not enough. In previous experiments, the space of hyperparameters is only between 0.0001 to 10. However, once we obtained a smaller range after using the first grid in figure 5.1, genetic algorithm was able to produce a better model than using another grid. The combination of grid search and GA demonstrates potential of finding optimal hyperparameters. Nevertheless, the improvement is very minimal in such a complex dataset. It's important to know that tuning hyperparameters may not cause drastic improvements as further improvements may need to be done on the model or the data itself.

The slight improvement from tuning hyperparameters can be demonstrated by the convergence curve when using GA and PSO alone.

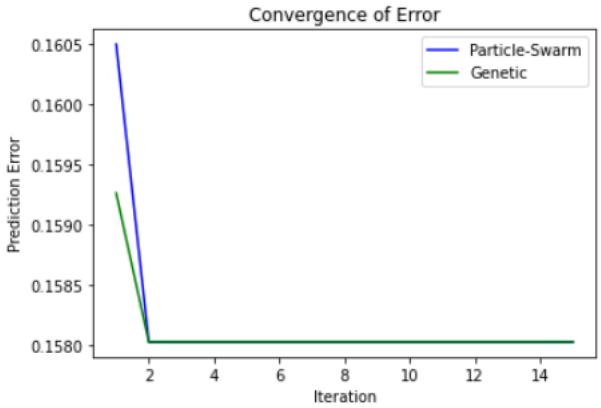

Figure 5.2: Convergence Curve of PSO and GA

Out of 15 iterations, both methods PSO and GA reached their minimum once in the second iteration. The improvements is neither significant, the error reduced from 15.93% to 15.88%. It's true that for some large and complex datasets, they are not very sensitive to small change in hyperparameters.

Chapter 6

Conclusion

In this paper, we compared a number of hyperparameter tuning methods on support vector machines in binary classification problems. In contrast to some of other works, this paper emphasized on the robustness of particle-swarm optimization and genetic algorithm. That is, will these two methods always obtain better hyperparameters than grid search given the same number of evaluations?

We observed that the performance of the methods depends on the dataset. In section 4.3, when the dataset is simple as make_circles, we observed that PSO and GA are able to obtain better hyperparameters than grid search with a sacrifice of run time. However, when the size and the complexity of the dataset increases, the advantage is diminishing. As shown in figure 4.1 and 4.2, dataset with size 100 gives more space for the model to be modified in achieving higher classification accuracy whereas when the size is 1000, two classes are more blended together, even both have an equal noise level at 0.1 when generating.

We also tested the effect of feature spaces in these methods' performance. A similar pattern was observed, when the dataset is small, the methods are not sensitive to the number of features and PSO, GA are able to outperform grid search in most of the cases. When the dataset is large, the performance are mixed with PSO and GA outperform grid search in some cases. We also found more variance in the performance from make_classifications than make_circles, it can be that make_classifications generates samples on random locations while make_circles always produce samples in a circular shape. This again reinforced that the performance of the methods is dependent on the dataset.

Lastly, to combine the advantage of grid search and these iterative methods, we proposed the warm-start genetic algorithm. We carried out the experiment on a real dataset that is more complex. Since the optimal hyperparameters for SVM could exist through the entire positive number space, we can use grid search to first filter a potential area for hyperparameters. Then we use this information to set the boundaries for the genetic algorithm to find the optimal hyperparameters. Grid search combined with genetic algorithm is able to outperform grid search and genetic algorithm alone with a slight higher prediction accuracy. However, when examine the convergence plot, we found that the algorithm stopped decreasing since the second iteration. Along with the heat-map we found the performance difference by different choices of hyperparameters is not significant.

Iterative methods like particle-swarm optimization and genetic algorithm are able to outperform traditional grid search method given the same number of evaluations in most of the cases. However, we could further benefit from PSO and GA in case when a model has too many hyperparameters (e.g. neural networks) where grid search could fail since the number of evaluations increase significantly with each additional hyperparameter. Nevertheless, PSO and GA demonstrates the potential of finding better hyperparameters given their versatility and generality.

Link to Github repo:

https://github.com/huanxiliu99/Eval-Hyperparameter-Tuning-Methods

Acknowledgments

Here I would like to appreciate the help and support I received from my advisor, Professor Ery Arias-Castro. It's been a great journey working along with him in the past quarters. I also want to thank to my peers in providing valuable insights and feedbacks.

Selected Bibliography Including Cited Works

[1] *Leaps - applied data science & ml certification program.*

[2] *Overfitting in machine learning: What it is and how to prevent it*, May 2020.

[3] J. Brownlee, *What is the difference between test and validation datasets?*, Aug 2020.

[4] J. Chen, H. Chen, Y. Huo, W. Gao, et al., *Application of svr models in stock index forecast based on different parameter search methods*, Open Journal of Statistics, 7 (2017), p. 194.

[5] L. Chen, *Support vector machine-simply explained*, Jan 2019.

[6] C. Cortes and V. Vapnik, *Support-vector networks*, Machine learning, 20 (1995), pp. 273–297.

[7] S. Gao, *Bio-Inspired Computational Algorithms and Their Applications*, BoD–Books on Demand, 2012.

[8] A. Ghaheri, S. Shoar, M. Naderan, and S. S. Hoseini, *The applications of genetic algorithms in medicine*, Oman medical journal, 30 (2015), p. 406.

[9] D. E. Goldberg and J. H. Holland, *Genetic algorithms and machine learning*, (1988).

[10] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*, Springer Science & Business Media, 2009.

[11] Hefinioanrhys, *Support vector machines with the mlr package: R-bloggers*, Oct 2019.

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

[13] J. Kennedy and R. Eberhart, *Particle swarm optimization*, in Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948 vol.4.

[14] Y. Li and Y. Zhang, *Hyper parameter estimation method with particle swarm optimization*, arXiv preprint arXiv:2011.11944, (2020).

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 12 (2011), pp. 2825–2830.

[16] B. D. Ripley, *Pattern recognition and neural networks*, Cambridge university press, 2007.

[17] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*, Cornell Aeronautical Laboratory, 1957.

[18] Y. Zhang, S. Wang, and G. Ji, *A comprehensive survey on particle swarm optimization algorithm and its applications*, Mathematical Problems in Engineering, 2015 (2015).