# APPLYING SUPPORT VECTOR MACHINES TO UNCERTAINTY QUANTIFICATION

ALEXANDER KORAN

*Date*: August 2022.

## 1. Abstract

Electrical components and systems are extremely prevalent in modern society. From the microchips within our smartphones to the large electricity transmission lines, these systems provide incredible technological capabilities that affect virtually every part of our lives.

As with every systems formed from numerous components, microchips have inherent variations that sum together to lead to uncertainty in the way that they are manufactured, characterized, and operated. As engineers, we hope to study these uncertainties, so that we can be confident in our ability to manufacture and operate these products. The need to do this has given rise to the field of study called Uncertainty Quantification.

One of the early and favored methods to simulate electrical circuits and systems is the Monte Carlo method. This operates by repeatedly picking values for the input parameters, calculating outputs, and gathering the results into a probability density curve. While this is a very effective approach, the issue that arises is that in many modern systems the complexity and number of uncertain parameters is so high that performing Monte-Carlo simulations becomes very computationally expensive. It is thus desirable to find alternative ways to simulate these intricate systems without taking too long or using too much computational power.

## 2. Introduction

One alternate approach to Monte Carlo simulations is to apply Machine Learning techniques to the area of Uncertainty Quantification. More specifically, we may use Support Vector Machines (SVMs) to try to predict our output distribution.

A Support Vector Machine attempts to find a hyperplane capable of separating and classifying a data set. SVMs can also be used for regression, in a sense finding the best fitting hyperplane. This is the basis for the approach outlined in this report.

In many cases, the data may not always be linearly separable so we may transform it through the means of a Kernel function. This effectively transforms the data into a preferred form for the regression to fit.

SVMs are a useful choice of model when accuracy is valued over extreme precision. The SVM used in this work incorporates a loss function that only penalizes sample points outside the "epsilon tube", that is, more than epsilon distance away from the prediction function. This scheme accelerates the time required to fit the data, albeit at the cost of some precision.

The current work also focused on developing, coding, and running a Least Squares SVM. This variant of the basic SVM uses the error between the predicted and actual values to efficiently find a good fit, and follows a method similar to least squares regression.

2.1. **Linear SVM.** To develop the linear SVM we begin by defining a set of training data

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n \subseteq \mathbb{R}^d \times \mathbb{R} \tag{1}$$

where $d$ is the dimension (number of parameters/variables) and $n$ is the number of data points.

We begin by describing the case of linear SVMs. This type of function takes the form

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b \tag{2}$$

where $\langle \cdot\,, \cdot \rangle$ denotes the standard inner product in $\mathbb{R}^d$. We want a small $w$ to keep the SVM model "flat". Hence we may write the optimization problem:

$$\begin{aligned} \min \quad & \frac{1}{2}||\boldsymbol{w}||^2 \\ \text{s.t.} \quad & y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b \leq \epsilon \\ & \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - y_i \leq \epsilon \end{aligned} \tag{3}$$

We are not actually guaranteed to find a function that can satisfy those constraints, so we introduce slack variables to allow extra margin for error. So now our optimization problem becomes

$$\begin{aligned} \min \quad & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^n (\xi_i + \xi_i^*) \\ \text{s.t.} \quad & y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b \leq \epsilon + \xi_i \\ & \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - y_i \leq \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \geq 0 \end{aligned} \tag{4}$$

To solve this, we construct the Lagrange function

$$\begin{aligned} L = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^n (\xi_i + \xi_i^*) - \sum_{i=1}^n (\eta_i \xi_i + \eta_i^* \xi_i^*) \\ - \sum_{i=1}^n \alpha_i (\epsilon + \xi_i - y_i + \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \\ - \sum_{i=1}^n \alpha_i^* (\epsilon + \xi_i^* + y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b) \end{aligned} \tag{5}$$

where $\alpha_i, \alpha_i^*, \eta_i, \eta_i^* \geq 0$ are Lagrange Multipliers. Now we set the gradient of our Lagrange function to zero:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^n (\alpha_i^* - \alpha_i) = 0 \tag{6}$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^n (\alpha_i - \alpha_i^*) x_i = 0 \tag{7}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \tag{8}$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0 \tag{9}$$

Substituting these into our original Lagrange function yields the dual optimization problem

$$\begin{aligned} \max \quad & \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_j-\alpha_j^*)\langle \boldsymbol{x}_i,\boldsymbol{x}_j\rangle \\ -\epsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*)+\sum_{i=1}^{n}y_i(\alpha_i-\alpha_i^*) \end{cases} \\ \text{s.t.} \quad & \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)=0 \text{ and } \alpha_i,\alpha_i^*\in[0,C] \end{aligned} \tag{10}$$

To solve this optimization problem, one may use a Quadratic Programming Toolbox, or an algorithm such as SMO (Sequential Minimal Optimization). Using Equation (7), we have

$$f(\boldsymbol{x})=\langle \boldsymbol{w},\boldsymbol{x}\rangle+b=\left\langle \sum_{i=1}^{n}(\alpha_i-\alpha_i^*)\boldsymbol{x}_i,\boldsymbol{x}\right\rangle+b=\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)\langle \boldsymbol{x}_i,\boldsymbol{x}\rangle+b \tag{11}$$

and thus we may plug in our solved values of $\alpha_i, \alpha_i^*, b$ to obtain our final SVM equation.

2.2. **Nonlinear SVM.** In order to use this method for data that might not be fitted linearly, we may map the input data $\boldsymbol{x}_i$ into some higher dimensional feature space through the use of some function $\phi : \mathbb{R}^d \to \mathcal{F}$. So our form is

$$f(\boldsymbol{x})=\langle \boldsymbol{w},\phi(\boldsymbol{x})\rangle+b \tag{12}$$

The calculations are nearly identical to the previous section. However, there is a slight computational drawback with this method. As the dimensionality of this feature space is increased, the computations become increasingly more time consuming.

We observe that in the final SVM form, we only deal with the inner product of $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$ (rather than $\phi$ by itself), meaning we may substitute $K(\boldsymbol{x}_i,\boldsymbol{x}_j)=\langle\phi(\boldsymbol{x}_i),\phi(\boldsymbol{x}_j)\rangle$. Then our final SVM function form is

$$f(\boldsymbol{x})=\sum_{i=1}^{n}(\alpha_i-\alpha_i^*)K(\boldsymbol{x}_i,\boldsymbol{x})+b \tag{13}$$

So, we may actually eliminate the use of our mapping $\phi$, and simply pick a suitable Kernel function for our SVM. This is known as the "Kernel trick". Some common Kernel functions include

- Linear: $K(\boldsymbol{x}_i,\boldsymbol{x})=\boldsymbol{x}_i^T\boldsymbol{x}$.
- Polynomial of order $q$: $K(\boldsymbol{x}_i,\boldsymbol{x})=(1+\boldsymbol{x}_i^T\boldsymbol{x})^q$.
- Gaussian Radial Basis Function (RBF): $K(\boldsymbol{x}_i,\boldsymbol{x})=\exp\left\{\frac{-||\boldsymbol{x}_i-\boldsymbol{x}||^2}{2\sigma^2}\right\}$.
- Sigmoid: $K(\boldsymbol{x}_i,\boldsymbol{x})=\tanh(\gamma\cdot\boldsymbol{x}_i^T\boldsymbol{x}+r)$.

## 3. Calculations

In another paper, they use a slightly different method which avoids using a Quadratic Programming Toolbox. They construct a similar optimization problem, with a few differences:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*} \quad & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + \frac{1}{2}b^2 + \frac{C}{2}\left[\sum_{i=1}^{n} w_p\xi_i^2 + \sum_{i=1}^{n} w_n\left(\xi_i^*\right)^2\right] \\ \text{s.t.} \quad & y_i - \boldsymbol{w}^T\phi(\boldsymbol{x}_i) - b \leq \epsilon + \xi_i \\ & \boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b - y_i \leq \epsilon + \xi_i^*, \quad i = 1, \ldots, n \end{aligned} \tag{14}$$

Note some of the differences:

- They include the bias term ($b$).
- They square the slack variables.
- They have separate weights for each type of slack variable ($w_p$ and $w_n$).

These changes will keep the math cleaner and preserve properties such as convexity and nonnegativity. Now, by the Representation Theorem, we may write our function in the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \beta_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b = \underbrace{\sum_{i=1}^{n} \beta_i \phi(\boldsymbol{x}_i)^T}_{\boldsymbol{w}^T} \phi(\boldsymbol{x}) + b \tag{15}$$

where $\beta_i$ and $b$ are regression parameters we will solve for. Since we know the general form of our SVM function is $y = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b$, we may conclude that

$$\boldsymbol{w} = \sum_{i=1}^{n} \beta_i \phi(\boldsymbol{x}) \tag{16}$$

Then

$$||\boldsymbol{w}||^2 = \boldsymbol{w}^T\boldsymbol{w} = \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i\beta_j\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i\beta_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\beta}^T\boldsymbol{K}\boldsymbol{\beta} \tag{17}$$

where

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} \qquad \boldsymbol{K} = \begin{bmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \tag{18}$$

We will use this substitution later to rewrite our optimization problem. Now, we introduce the asymmetric loss function, defined as

$$V_\epsilon(x) = \begin{cases} w_p(x-\epsilon)^2 & \text{if } x > \epsilon \\ 0 & \text{if } |x| \leq \epsilon \\ w_n(x+\epsilon)^2 & \text{if } x < -\epsilon \end{cases} \tag{19}$$

where $x \in \mathbb{R}$ is some scalar. When we plug in the error term $y_i - f(\boldsymbol{x}_i)$, we output the corresponding slack variable term.

Using this loss function and some substitutions, we may rewrite our optimization problem as follows:

$$\min_{\boldsymbol{w},b} F(\boldsymbol{w},b) = \frac{1}{2}||\boldsymbol{w}||^2 + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n} V_\epsilon\left(y_i - f(\boldsymbol{x}_i)\right) \tag{20}$$

$$\min_{\boldsymbol{\beta},b} F(\boldsymbol{\beta},b) = \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{K}\boldsymbol{\beta} + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n} V_\epsilon\left(y_i - f(\boldsymbol{x}_i)\right) \tag{21}$$

$$\min_{\boldsymbol{\beta},b} F(\boldsymbol{\beta},b) = \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{K}\boldsymbol{\beta} + \frac{1}{2}b^2 + \frac{C}{2}\sum_{i=1}^{n} (y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b - e_i)^2 d_i \tag{22}$$

where $d_i$ and $e_i$ are defined as

$$d_i = \begin{cases} w_p & \text{if } y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b > \epsilon \\ 0 & \text{if } |y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b| \leq \epsilon \\ w_n & \text{if } y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b < -\epsilon \end{cases} \tag{23}$$

$$e_i = \begin{cases} \epsilon & \text{if } y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b > \epsilon \\ 0 & \text{if } |y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b| \leq \epsilon \\ -\epsilon & \text{if } y_i - \boldsymbol{K}_i\boldsymbol{\beta} - b < -\epsilon \end{cases} \tag{24}$$

To simplify, we can write $F(\boldsymbol{\beta},b)$ in matrix form:

$$F(\boldsymbol{\beta},b) = \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{K}\boldsymbol{\beta} + \frac{1}{2}b^2 + \frac{C}{2}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e})^T\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e}) \tag{25}$$

where

$$\boldsymbol{e} = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \boldsymbol{D} = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{bmatrix} \tag{26}$$

To minimize $F(\boldsymbol{\beta},b)$, we take the gradient with respect to $\boldsymbol{\beta}$ and set it to zero:

$$\frac{\partial F}{\partial \boldsymbol{\beta}} = \boldsymbol{K}\boldsymbol{\beta} - C\boldsymbol{K}^T\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e}) = \mathbf{0}_n \tag{27}$$

Since $\boldsymbol{K}$ is invertible and symmetric, we have

$$\boldsymbol{K}\boldsymbol{\beta} - C\underbrace{\boldsymbol{K}^T}_{=\boldsymbol{K}}\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e}) = \mathbf{0} \tag{28}$$

$$\boldsymbol{K}^{-1}\left[\boldsymbol{K}\boldsymbol{\beta} - C\boldsymbol{K}\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e})\right] = \boldsymbol{K}^{-1}\mathbf{0} \tag{29}$$

$$\boldsymbol{I}\boldsymbol{\beta} - C\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e}) = \mathbf{0} \tag{30}$$

$$\frac{\boldsymbol{I}\boldsymbol{\beta}}{C} - \boldsymbol{D}(\boldsymbol{y} - \boldsymbol{e}) + \boldsymbol{D}(\boldsymbol{K}\boldsymbol{\beta} + b\mathbf{1}) = \mathbf{0} \tag{31}$$

$$\frac{\boldsymbol{I}\boldsymbol{\beta}}{C} + \boldsymbol{D}\boldsymbol{K}\boldsymbol{\beta} + \boldsymbol{D}b\mathbf{1} = \boldsymbol{D}(\boldsymbol{y} - \boldsymbol{e}) \tag{32}$$

$$\left(\frac{\boldsymbol{I}}{C} + \boldsymbol{D}\boldsymbol{K}\right)\boldsymbol{\beta} + \boldsymbol{D}\mathbf{1}b = \boldsymbol{D}(\boldsymbol{y} - \boldsymbol{e}) \tag{33}$$

Similarly, we take the gradient with respect to $b$ and set it to zero:

$$\frac{\partial F}{\partial b} = b - C\mathbf{1}^T\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e}) \tag{34}$$

$$b - C\mathbf{1}'\boldsymbol{D}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\beta} - b\mathbf{1} - \boldsymbol{e}) = 0 \tag{35}$$

$$\frac{b}{C} - \mathbf{1}'\boldsymbol{D}(\boldsymbol{y}-\boldsymbol{e}) + \mathbf{1}'\boldsymbol{D}\boldsymbol{K}\boldsymbol{\beta} + \mathbf{1}'\boldsymbol{D}b\mathbf{1} = 0 \tag{36}$$

$$\mathbf{1}'\boldsymbol{D}\boldsymbol{K}\boldsymbol{\beta} + \left(\mathbf{1}'\boldsymbol{D}\mathbf{1} + \frac{1}{C}\right) b = \mathbf{1}'\boldsymbol{D}(\boldsymbol{y}-\boldsymbol{e}) \tag{37}$$

Writing equations (33) and (37) in a compact form gives

$$\begin{bmatrix} \frac{\boldsymbol{I}}{C} + \boldsymbol{D}\boldsymbol{K} & \boldsymbol{D}\mathbf{1} \\ \mathbf{1}^T\boldsymbol{D}\boldsymbol{K} & \mathbf{1}^T\boldsymbol{D}\mathbf{1} + \frac{1}{C} \end{bmatrix} \begin{bmatrix} \boldsymbol{\beta} \\ b \end{bmatrix} = \begin{bmatrix} \boldsymbol{D}(\boldsymbol{y}-\boldsymbol{e}) \\ \mathbf{1}^T\boldsymbol{D}(\boldsymbol{y}-\boldsymbol{e}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} \\ \mathbf{1}^T \end{bmatrix} \boldsymbol{D}(\boldsymbol{y}-\boldsymbol{e}) \tag{38}$$

This system can be solved not directly, but by using an iterative procedure:

(0) Initialization: choose a starting point $\boldsymbol{\beta}^0 \in \mathbb{R}^p$ and $b^0 \in \mathbb{R}$, set the tolerance level $\epsilon$, the maximum iteration number $M$. Set the iteration number $t = 0$.

(1) Using the current $\boldsymbol{\beta}^t$ and $b^t$, apply Eq. 9 and 10 to calculate $d_i$ and $e_i$, and form the matrix $\mathbf{D}$ and vector $\mathbf{e}$.

(2) Update $\boldsymbol{\beta}$ and $b$ as

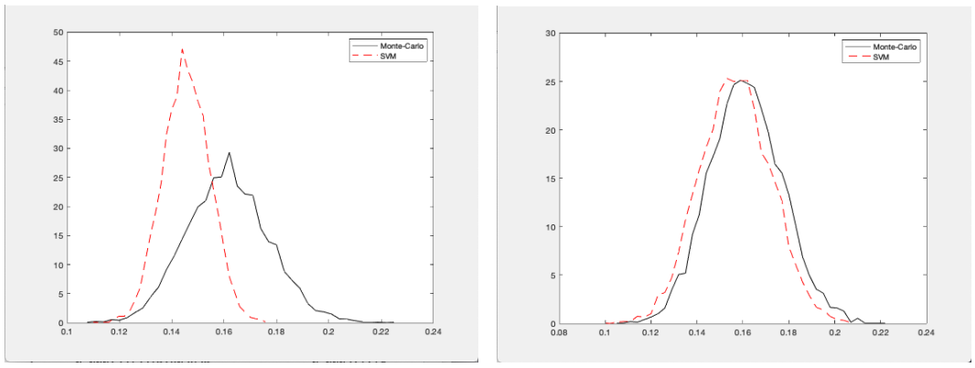
$$\begin{bmatrix} \boldsymbol{\beta}^{t+1} \\ b^{t+1} \end{bmatrix} = \begin{bmatrix} \frac{\boldsymbol{I}}{C} + \boldsymbol{D}\boldsymbol{K} & \boldsymbol{D}\mathbf{1} \\ \mathbf{1}^T\boldsymbol{D}\boldsymbol{K} & \mathbf{1}^T\boldsymbol{D}\mathbf{1} + \frac{1}{C} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{I} \\ \mathbf{1}^T \end{bmatrix} \boldsymbol{D}(\boldsymbol{y}-\boldsymbol{e})$$

(3) Stop if $||\boldsymbol{\beta}^{t+1} - \boldsymbol{\beta}^t|| + |b^{t+1} - b^t| < \epsilon$ or $t = M$; otherwise, set $t = t + 1$ and go back to step 1.

(4) Return the current $\boldsymbol{\beta}$ and $b$.

So after solving for $\boldsymbol{\beta}$ and $b$, we plug them in to obtain our final SVM function

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \beta_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b \tag{39}$$




(A) 10 Training Samples (B) 40 Training Samples

(C) 400 Training Samples (D) 4000 Training Samples

FIGURE 1. Relative fit of SVM to Monte Carlo model as a function of training sample size.

## 4. RESULTS

4.1. **SVM Modelling Overview.** An SVM was modelled in MATLAB. Input data was generated using Latin Hypercube Sampling. This data is then used as the input parameters for an RLC circuit (Wang, Khazaka). The RLC circuit program then calculates an output voltage for all input parameters. This output is then used for training and validation by the SVM.

The SVM was run using a linear kernel, a threshold value of $1e-5$, and a validation set size of 40. The value of epsilon was chosen to be $1e-3$, $C=1$, and $w_p = w_n = 1$. Figure 1 shows the Probability Distribution Function (PDF) of the SVM as a function of training set size, compared to the PDF of a Monte Carlo simulation with 5000 points. It is clear from the figures that the curves overlap quite well for training set sizes of approximately 400 or larger.

The data generated demonstrate that the SVM can effectively model a fixed data set smaller than the outputs required by Monte Carlo. The following sections will begin to evaluate the sensitivity of the parameters used in the SVM on modelling effectiveness.

4.2. **Single Variable Effect Analysis.** In this section, we plot the resulting Mean Squared Error (MSE) of our SVM model predictions while varying the

following parameters: the number of training samples, $\epsilon$, $C$, $w_p$, and $w_n$. The data is graphed in Figures 2-6 on a log-log scale to better illustrate the trend.

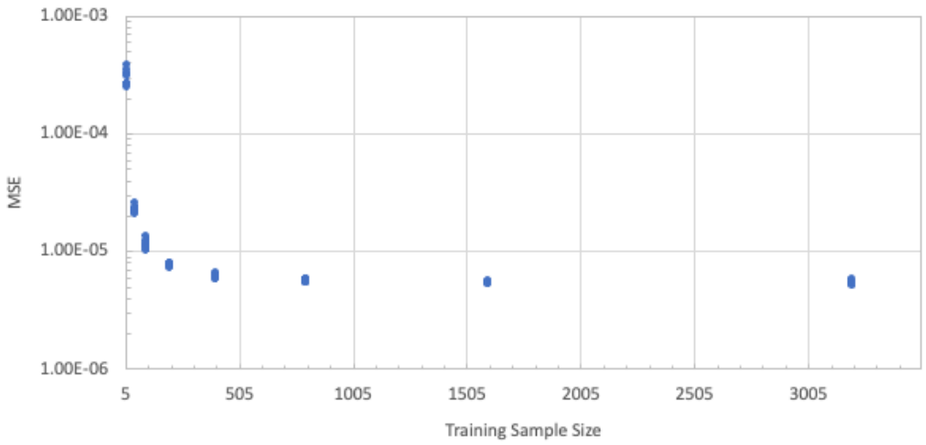



FIGURE 2. Effect of Training Set Size on MSE

Here in Figure 2 we see how varying the number of training samples changes the MSE. Naturally, it makes sense that using more training data will reduce the error in our model. It is important to note that the MSE levels off around $5 \cdot 10^{-6}$, meaning that increasing the number of training samples further will at some point not change the MSE of our model.

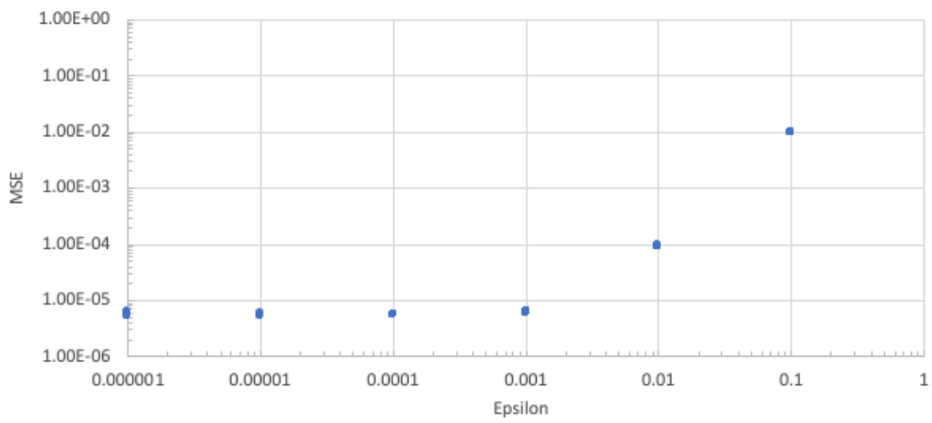



FIGURE 3. Effect of Epsilon on MSE

In Figure 3, we see how increasing $\epsilon$ increases the MSE. This makes sense, as widening the $\epsilon$-tube will allow more data to be inside, but the overall accuracy of the SVM will decrease, hence the error will increase.

An interesting note is that as we decrease $\epsilon$, the error decreases, but eventually levels off. A hypothesis for this is that the slack variables $\xi_i, \xi_i^*$ are forced to be larger, and account for the data outside the $\epsilon$-tube.

In Figure 4, we see how $C$ affects the MSE. As $C$ increases, the MSE decreases, and eventually levels off when $C > 1$. Several of the calculations have $\frac{1}{C}$ in them, so those terms shrink to zero as $C$ increases, and the rest stays constant.

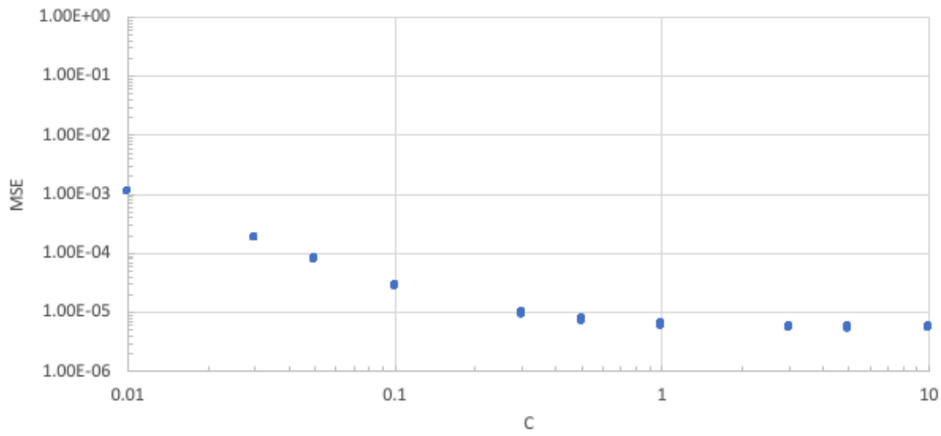



FIGURE 4. Effect of $C$ on MSE

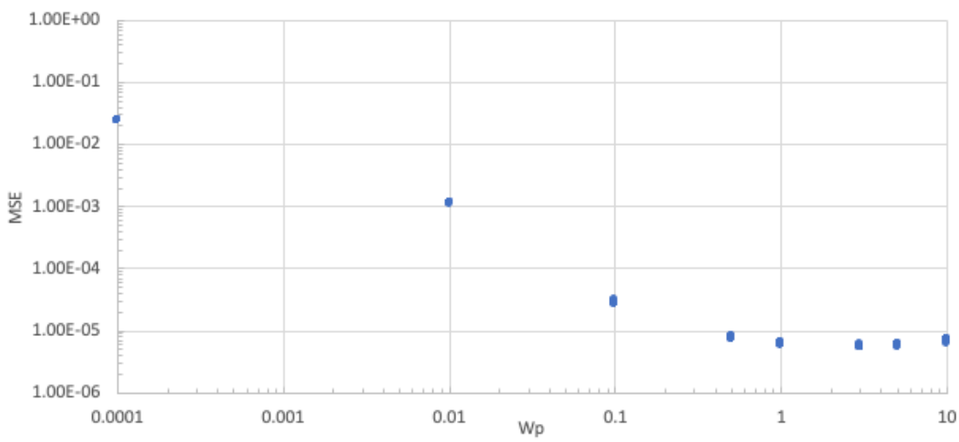



FIGURE 5. Effect of $w_p$ on MSE

Figures 5 and 6 show the effect of $w_p$ and $w_n$, the asymmetric slack variables weights, on the MSE. The $w_p$ weight seems to have a large effect as it is decreased, increasing the MSE significantly. In contrast, $w_n$ does not have a very large effect. As $w_n$ becomes very large, the error finally begins to grow, as well as its variability (as can be seen in Figure 6).

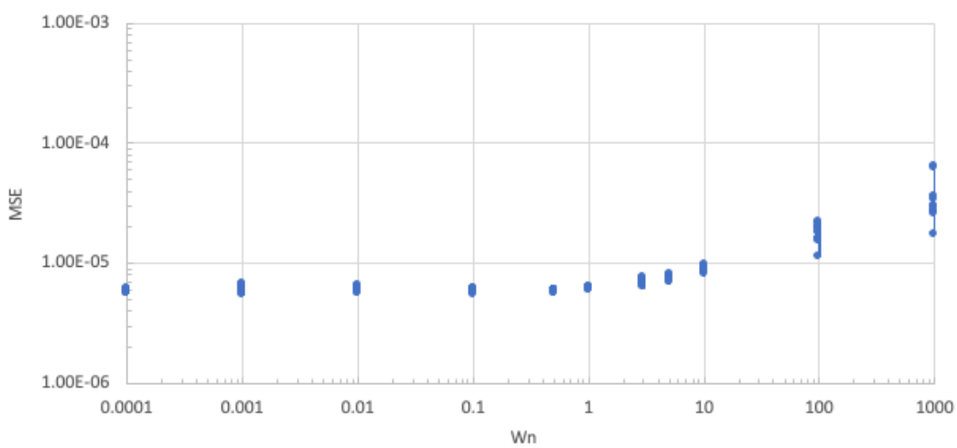



FIGURE 6. Effect of $w_n$ on MSE

4.3. **Multivariable Effect Analysis.** Additional simulations were carried out to attempt to investigate the interaction effects of the various simulation parameters. A design of experiments was created using the JMP statistical software package. The DOE used is included as Table 1 in the Appendix. The JMP statistical software was also used to model the interaction effects.

A first attempt at modelling produced a fit with an $R^2$ value of 0.41. Careful analysis of the data from Section 4.2 indicated that it follows a logarithmic scale. The DOE data was thus converted to logarithmic scale, and inputted once again to the JMP modelling routine. A second order model with the newly inputted logarithmic data resulted in a much higher $R^2$ value of 0.92.

A plot of the different interactive effects of the variables is show in Figure 7. In this plot, all of the parameters were selected to minimize MSE. As expected, the lowest values of MSE are obtained with the highest number of training samples. It is clearly undesirable, however, to use excessively large amounts of training data, as we may not have a surplus of data points.

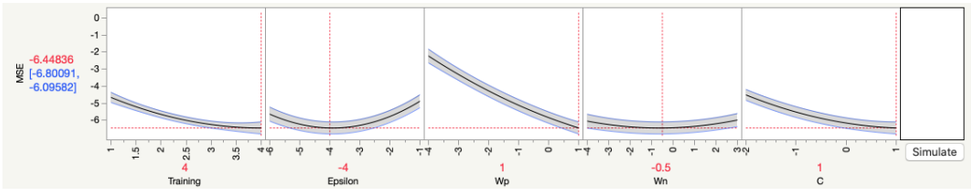



FIGURE 7. Interactive Effect of Simulation Parameters on MSE

Excessive amounts of training data is also expected to increase the computational load of training the SVM. Simulations were also carried out to approximate the computational time required to train the SVM as a function of the modelling parameters. Figure 8 highlights that computational time increases with the number of training samples, but is essentially unaffected by the other parameters.

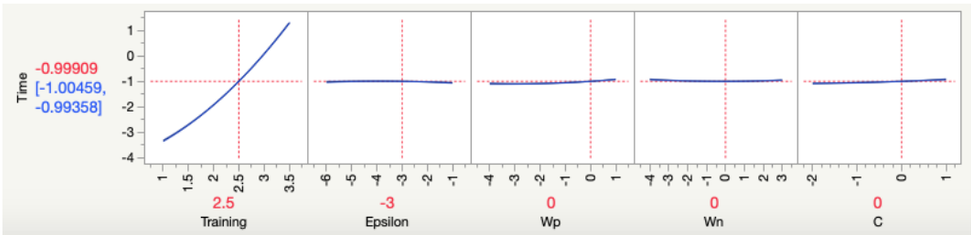



FIGURE 8. Interactive Effect of Simulation Parameters on SVM Computational Time

It should be noted that in this analysis, computational load was carried out by measuring the time required to train the SVM. The resulting time calculated is thus a function of the computational power of the system used to run the calculations. The magnitude of the effect of training size on time, as shown in Figure 8, should thus only be used to show the direction of the trend,

rather than an absolute measurement of computational load. There are better ways to calculate computational load, but this was not part of the scope of the project.

A more desired approach to optimizing the SVM would be to reduce the number of training samples, but maintain all of the other modelling parameters that minimize the MSE. Figure 9 highlights this.

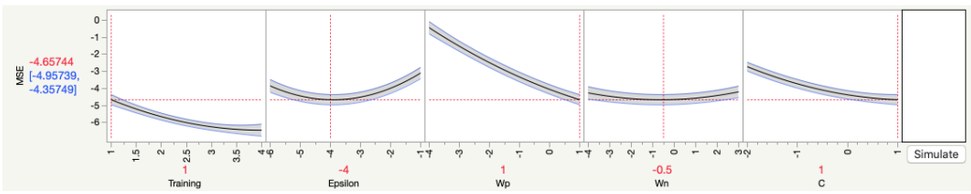



FIGURE 9. Interactive Effect of Simulation Parameters on MSE

The difference in approach between Figure 7 and 9 result in an MSE of $3.56 \times 10^{-7}$ and $2.20 \times 10^{-5}$, respectively. While the difference is significant, the error associated with only 10 training samples is surprisingly good.

4.4. **Effect of Kernel Type.** It is to be expected that the kernel function used in the simulation will affect both the precision and calculation load of the SVM. Two additional kernel functions were evaluated to investigate this effect.

A polynomial kernel function was evaluated with degrees varying between 1, 2, and 3. As in previous cases, a DOE was run and the data was evaluated using JMP. Figure 10 shows the output. The model reveals that it is possible to effectively use the polynomial kernel type in SVM modelling. The parameter $C$ provides the strongest response, minimizing MSE as we progress to higher values of $C$. Reviewing the data, it would've been interesting to explore even higher values of $C$ than called for in the DOE; this can be part of a future work.

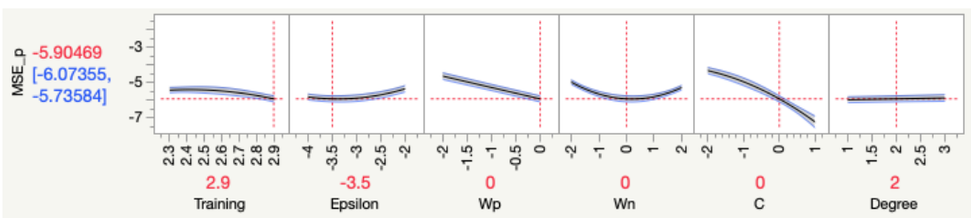



FIGURE 10. Interactive Effect of Simulation Parameters on MSE with Polynomial Kernel Type

As with the linear kernel, from Figure 11, it can be seen that the time required to train an SVM utilising a polynomial kernel increases with training set size. It is relatively unaffected by all other variables, including the degree of the polynomial.

A Gaussian RBF kernel function was evaluated with the $\sigma$ parameter varying between 1 and 20. As in previous cases, the data was evaluated using JMP.

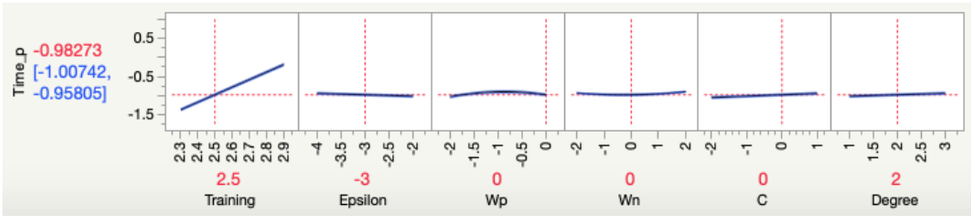



FIGURE 11. Interactive Effect of Simulation Parameters on Time with Polynomial Kernel Type

Figure 12 shows the output. Once again, the model reveals that it is possible to effectively use the Gaussian RBF kernel type in SVM modelling. The parameter $C$ provides the strongest response, minimizing MSE as we progress to higher values of $C$. Yet again, after reviewing the data, it would have been interesting to explore even higher values of $C$ than called for in the DOE; this can be part of a future work.

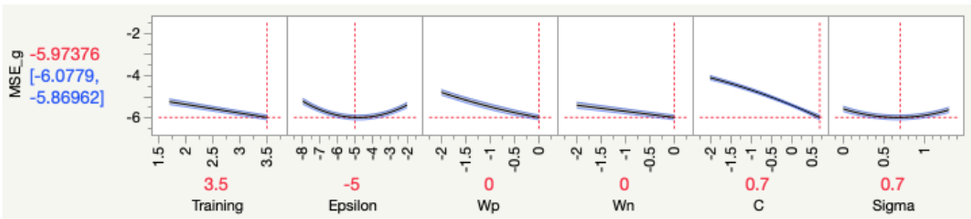



FIGURE 12. Interactive Effect of Simulation Parameters on MSE with Gaussian RBF Kernel Type

As with the linear and polynomial kernels, from Figure 13, it can be seen that the time required to train an SVM utilising a Gaussian RBF kernel increases with training set size. It is relatively unaffected by all other variables, including the $\sigma$ parameter.

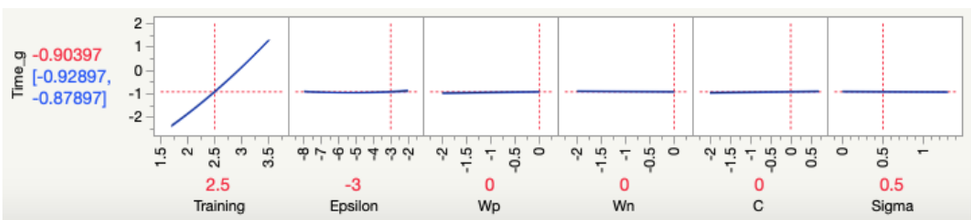



FIGURE 13. Interactive Effect of Simulation Parameters on Time with Gaussian RBF Kernel Type

4.5. **Reduction of Training Data.** Additional work was done to reduce the amount of required training data. In the area of Uncertainty Quantification, we do not necessarily have sufficient data available. For this reason, experiments were designed and executed to determine how few samples we can train on to still get accurate results.

We started with 10 training samples and increments of 10 for the number of samples in each iteration. We used a Validation Set to calculate the mean

squared error between the predicted and actual values. Once this error dropped beneath a tolerance, we stopped the iterations and used this number of training samples. Finally, we tested this model on a larger test set, against a Monte Carlo simulation with 5000 data points.

To evaluate how well the model works in training data reduction mode, we first have to understand the model response to the variation in the raw data created by the Latin Hypercube Sampling. This can be accomplished by running the SVM repeatedly to determine its variability.

Figure 14 illustrates the minimum sample size required to meet a threshold error value of $1e-5$ over a number of repeated runs. The Y-axis, labelled "Count", specifies the number of times that the process converged for a fixed training set size, per X-axis. The curve labelled 500 illustrates this illustrates this total count when the SVM is run 500 times, with a distinct training set each time. The figure shows the difference in the "Count" as a function of how many times the process is run. From the figure (and through normalizing the data), it can be seen that repeating the process 1000 times or more yields consistent results. Subsequent analyses will thus be carried out using SVM repetition of 1000 trials.

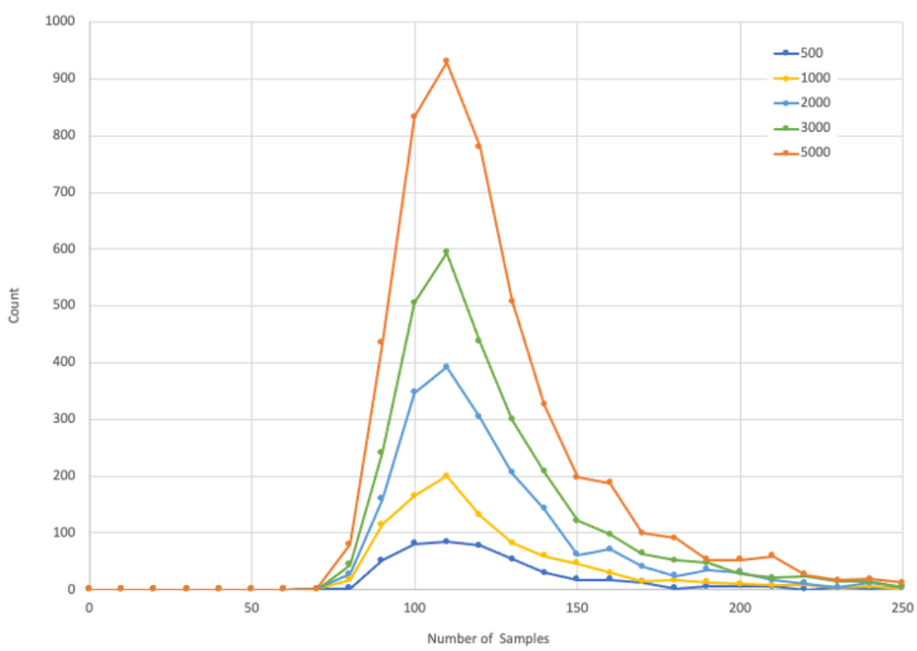



FIGURE 14. Interactive Effect of Simulation Parameters on Time with Gaussian RBF Kernel Type

Now that we determined the minimum number of SVM repetitions required to understand modelling variability, we can move on to investigating the effect of convergence threshold. Figure 15 once again highlights the total "Count" as a function of convergence threshold (i.e. $\epsilon$ in the iterative procedure in Section 3). Curves with high total counts and narrower spread are found for

thresholds from $1e-7$ to $1e-4$. The threshold of $1e-5$ used in many of the calculations in this report seems to fall in a desirable region.

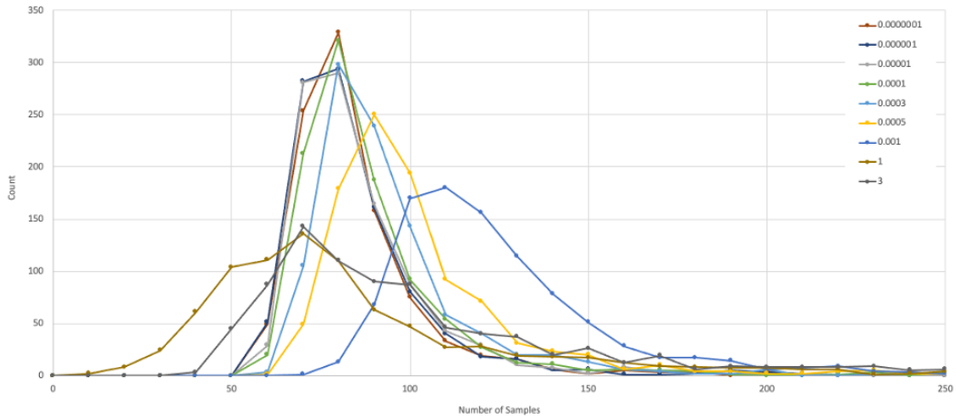



FIGURE 15. Interactive Effect of Simulation Parameters on Time with Gaussian RBF Kernel Type

It is generally preferable to use smaller validation sets to use less data overall. Larger validation sets do tend to lead to higher modelling precision. In Figure 16 we evaluated the effect of validation set size on the total "Count" of convergence (for SVM repeats of 1000 and threshold error of $1e-5$). From the figure, it can be seen that larger validation sets lead to curves with higher total counts and narrower spread. Smaller validation sets can lead to convergence at lower training set sizes because the validation set is likely too small to full access the performance of the trained SVM.

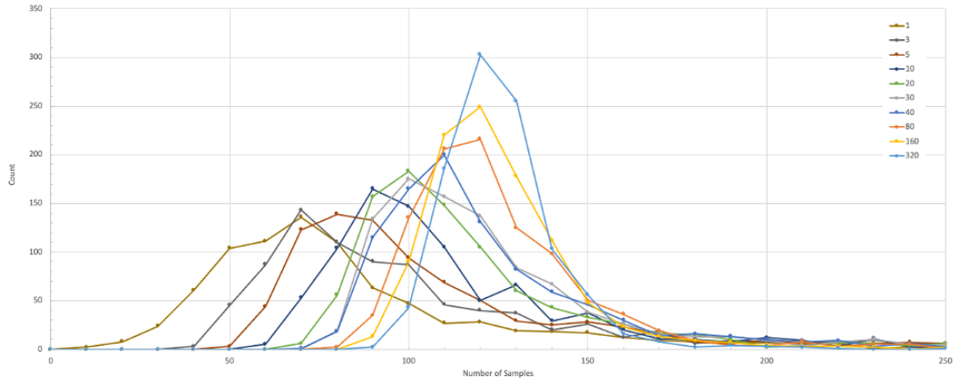



FIGURE 16. Interactive Effect of Simulation Parameters on Time with Gaussian RBF Kernel Type

Significantly more work is needed to expand on the data generated and to find reasonable ways to determine effective threshold error and validation set size levels. A hypothesis to be evaluated would be to use full width half max (FWHM) values of these curves for evaluation purposes.

It is interesting to note that some iterations never converged, even when we continually append new points to our training data set. In other words, the

MSE converged to a value that was above the error convergence threshold. It is unclear why this occurs for some trials; it is likely due to the samples obtained from Latin Hypercube Sampling. This is certainly an area for further investigation.

## 5. Future Works

One of the paths to continue this work would be to improve the accuracy of the SVM model through better hyperparameter tuning methods, such as Simulated Annealing and Optimization techniques for initial values. The other path would be to work on further reducing the number of training samples needed, perhaps by using a different version of SVM or with a better sampling technique.

## 6. Appendix

In Table 1, we show the Design of Experiments (DOE) generated by JMP and carried out in MATLAB.

| Run | # Train | Epsilon | Wp | Wn | C |
|---|---|---|---|---|---|
| 1 | 800 | 0.0001 | 0.01 | 100 | 1 |
| 2 | 800 | 0.01 | 0.01 | 0.01 | 0.01 |
| 3 | 200 | 0.01 | 0.01 | 100 | 0.01 |
| 4 | 800 | 0.0001 | 1 | 0.01 | 0.01 |
| 5 | 200 | 0.01 | 1 | 100 | 0.01 |
| 6 | 200 | 0.0001 | 0.01 | 100 | 1 |
| 7 | 200 | 0.01 | 1 | 100 | 1 |
| 8 | 200 | 0.0001 | 1 | 100 | 1 |
| 9 | 800 | 0.0001 | 0.01 | 0.01 | 1 |
| 10 | 200 | 0.0001 | 0.01 | 0.01 | 1 |
| 11 | 200 | 0.0001 | 1 | 100 | 0.01 |
| 12 | 800 | 0.0001 | 1 | 100 | 0.01 |
| 13 | 800 | 0.0001 | 0.01 | 100 | 0.01 |
| 14 | 800 | 0.01 | 1 | 100 | 1 |
| 15 | 200 | 0.0001 | 1 | 0.01 | 0.01 |
| 16 | 200 | 0.0001 | 0.01 | 100 | 0.01 |
| 17 | 800 | 0.0001 | 1 | 100 | 1 |
| 18 | 800 | 0.01 | 1 | 0.01 | 0.01 |
| 19 | 800 | 0.01 | 0.01 | 100 | 1 |
| 20 | 200 | 0.0001 | 0.01 | 0.01 | 0.01 |
| 21 | 200 | 0.01 | 1 | 0.01 | 1 |
| 22 | 200 | 0.01 | 1 | 0.01 | 0.01 |
| 23 | 400 | 0.001 | 0.1 | 100 | 0.1 |
| 24 | 800 | 0.01 | 1 | 100 | 0.01 |
| 25 | 800 | 0.0001 | 1 | 0.01 | 1 |
| 26 | 400 | 0.0001 | 1 | 0.01 | 0.01 |
| 27 | 800 | 0.01 | 0.01 | 0.01 | 1 |
| 28 | 200 | 0.01 | 0.01 | 0.01 | 1 |
| 29 | 800 | 0.01 | 1 | 0.01 | 1 |
| 30 | 800 | 0.0001 | 0.01 | 0.01 | 0.01 |
| 31 | 200 | 0.01 | 0.01 | 0.01 | 0.01 |
| 32 | 200 | 0.0001 | 1 | 0.01 | 1 |
| 33 | 200 | 0.01 | 0.01 | 100 | 1 |
| 34 | 400 | 0.01 | 0.01 | 0.01 | 0.01 |
| 35 | 400 | 0.0001 | 0.01 | 0.01 | 1 |
| 36 | 800 | 0.01 | 0.01 | 100 | 0.01 |

Table 1. DOE Table generated by JMP

## 7. Acknowledgements

I would like to thank Professor Roni Khazaka, Karan Sidhu, and William Wang for their continued guidance and support of this project.

## 8. References


- A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," Statist. Comput., vol. 14, no. 3, pp. 199–222, 2004.
- Songfeng Zheng (2021) Iteratively reweighted least square for asymmetric L2- Loss support vector regression, Communications in Statistics - Simulation and Computation.
- R. Trinchero, M. Larbi, H. M. Torun, F. G. Canavero and M. Swaminathan, "Machine Learning and Uncertainty Quantification for Surrogate Models of Integrated Devices With a Large Number of Parameters," in IEEE Access, vol. 7, pp. 4056-4066, 2019, doi: 10.1109/ACCESS.2018.2888903.