A Study and Implementation of a Support Vector Machine

2019 - Samuel Sheehy

1 INTRODUCTION

A Support Vector Machine (SVM) is a popular supervised machine learning technique for performing binary classification of data. By taking labelled training data, it can then predict the type of new unlabelled data points based on their attributes. In its essence, an SVM is an optimisation of a system subject to a set of constraints. The plethora of optimisation algorithms that exist give many options for resolving the SVM objective function. Furthermore, an SVM can be used in classification problems with more than two labels by reimplementing it several times on the set and changing the binary labelling of each class.

This report briefly examines the application of a simplified Sequential Minimal Optimisation (SMO) algorithm for the creation of a SMO and the subsequent treatment of the popular *Iris* dataset. In this context, a simulation study is proposed and implemented, which assesses the impact of using two different kernel functions on the performance of the SMO method.

2 BACKGROUND THEORY

2.1 INTRODUCTION TO STATE VECTOR MACHINES

In its simplest form, a SVM draws a line through a dataset separating two groups of data neatly as exemplified in Figure 1. New points with unknown labels are classified according to which side of line they fall. In general, regardless of the number of dimensions, this line is in fact a hyperplane, but its principal of operation is the same: the hyperplane determines where the data types can be separated, and new points are classified according to their placement with respect to this hyperplane.

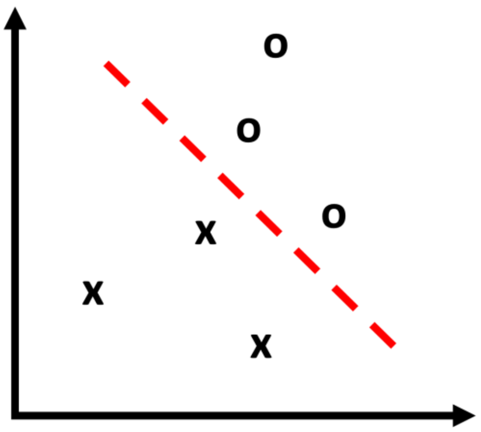


Figure 1 – An SVM draws a plane separating plane between two types of data

A hyperplane is defined by two parameters: its normal vector $\mathbf{w}$ and a constant b used to calculate the distance between the hyperplane and the origin, such that for every point $\mathbf{x}$ on the hyperplane we have:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

By $\mathbf{x}_i$ we denote a point i of the dataset. By representing the binary labels y_i of the data by either $+1$ or -1, $\mathbf{w}$ and b can be chosen so that the points closest to the hyperplane on either side can be represented by the following two expressions (we choose the values for $y_i = \pm 1$ for simplicity but in reality, any value instead of ± 1 would lead to the same solution):

$$\mathbf{w} \cdot \mathbf{x}_i + b = +1, \qquad \text{for } y_i = +1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b = -1, \qquad \text{for } y_i = -1$$

And the rest of the dataset can be described by these following two equations:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1, \qquad \text{for } y_i = +1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1, \qquad \text{for } y_i = -1$$

The two are combined into one through the following:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \; \forall i$$

In order to determine $\mathbf{w}$, we aim to maximise the margin $\frac{1}{||\mathbf{w}||}$ which is done by minimising $||\mathbf{w}||$ subject to the preceding inequality. To facilitate the optimisation implementation, we come to an equivalent form of the base problem which is the foundation for developing the SVM algorithm:

$$\min \frac{1}{2} ||\mathbf{w}||^2, \qquad s.t, \qquad y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \geq 0 \quad \forall_i$$

2.2 Solution of the Problem through the Dual Lagrangian

The original from of the problem is not straightforward nor efficient to solve directly. For this reason, the standard approach is to use the Lagrangian method to combine the objective function and the constraint into a single function, referred to us as the Primal problem that wants to be minimised for $\mathbf{w}$ and b:

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \boldsymbol{\alpha}[y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 \;\forall_i], \qquad \alpha_i \geq 0 \;\forall_i$$

Giving the following:

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{L} \alpha_i y_i(\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^{L} \alpha_i$$

The minima of L_P is achieved for $\mathbf{w}$ and b when their respective derivatives are zero. The derivatives then allow for the expression of $\mathbf{w}$ in terms of α_i, y_i and $\mathbf{x}_i$:

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \;\Rightarrow\; \mathbf{w} = \sum_{i=1}^{L} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L_P}{\partial b} = 0 \;\Rightarrow\; \sum_{i=1}^{L} \alpha_i y_i = 0$$

Substituting these results back into the expression for L_P gives the dual form of the primary, which is fully dependent on $\boldsymbol{\alpha}$ and which needs to be maximised in order to be consistent with the original problem:

$$L_D \equiv \sum_{i=1}^{L} \alpha_i - \frac{1}{2}\sum_{ij} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j\,, \qquad \alpha_i \geq 0 \;\forall_i, \qquad \sum_{i=1}^{L} \alpha_i y_i = 0$$

By setting the constant value matrix $\mathbf{H}$ such that $H_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$, the optimisation problem can be expressed as a convex quadratic optimisation problem to determine $\boldsymbol{\alpha}$, which can then be used to find **$\mathbf{w}$:**

$$\max_{\alpha} \sum_{i=1}^{L} \alpha_i - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha}, \qquad \alpha_i \geq 0 \;\; \forall_i, \qquad \sum_{i=1}^{L} \alpha_i y_i = 0$$

The optimal solution $\boldsymbol{\alpha}$ for L_D will have many elements α_i that are zero and with rest being finite, non-zero positive values. The non-zero values α_i correspond to the points $\mathbf{x}_i$ that are Support Vectors: that means to say, these points are sufficient to define the hyperplane separating the two classes of data.

In addition, these points that are Support Vectors which we will designate now $\mathbf{x}_S$ satisfy the equation:

$$y_S(\mathbf{x}_S \cdot \mathbf{w} + b) = 1$$

Using the term for $\mathbf{w}$ derived earlier and averaging all the values of b that are derived this leads to the expression, where N_S is the number of elements in S:

$$b = \frac{1}{N_S}\sum_{i \in S}\left(y_i - \sum_{j \in S} \alpha_j y_j x_j \cdot x_i\right)$$

2.3 Hard vs. Soft Margins

Up to this point, the data points were considered to be completely separable: that is to say it was possible to draw a straight line (or more generally a hyperplane) that could divide the two classes of points neatly. In practice however, this is not always possible because these data points are intermixed (due either to misclassifications or to processes where the attributes of two classes can overlap). The approach to resolve this issue is exemplified in Figure 2.

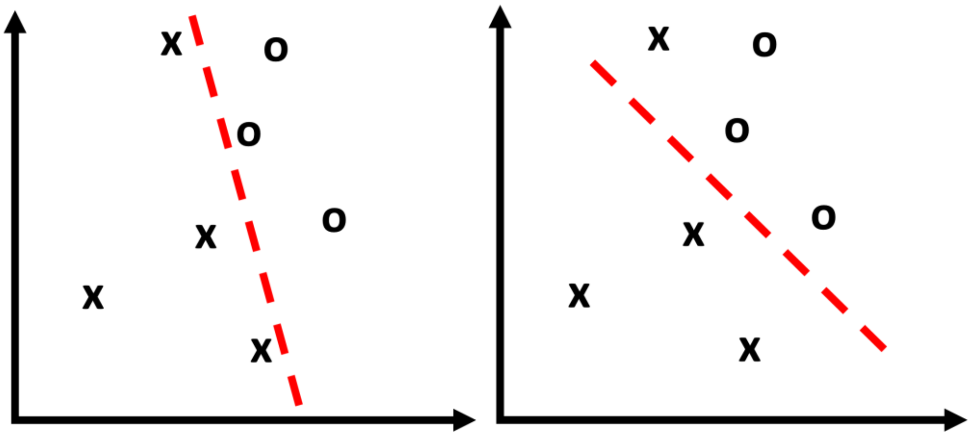


Figure 2 – Left: *A separating plane with no penalty parameter.*
Right: *A separating plane with a penalty parameter*

The approach to implement this is to apply a penalty to a point on the 'incorrect' side of the margin proportional to its distance from it. Returning to the initial statements of the classification problem, this is done with a slack variable that is introduced to allow for some variability in the hyperplane margin. These give the following equations:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq +1 - \xi, \qquad \text{for } y_i = +1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b \leq -1 + \xi\,, \qquad \text{for } y_i = -1$$

With $\xi \geq 0\ \forall_i$.

In a similar manner as before, these lead to a variation of the foundation problem:

$$\min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{L} \xi_i\,, \qquad s.t, \qquad y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0 \ \ \forall_i$$

Here the parameter C is a real number constant that determines the weight of the penalty term. The Lagrangian is reformulated as follows with a new penalty weight parameter $\mu_i \geq 0$:

$$L_P \equiv \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{L} \xi_i - \sum_{i=1}^{L} \alpha_i [y_i(\mathbf{x}_i \cdot \mathbf{w} + b) - 1 + \xi_i] - \sum_{i=1}^{L} \mu_i \xi_i$$

Differentiating with respect to $\mathbf{w}$, b and ξ_i and setting these to zero, the same two partial derivatives as earlier appear as well as the following one and the relationship we can conclude from it:

$$\frac{\partial L_P}{\partial \xi_i} = 0 \ \Rightarrow \ C = \alpha_i + \mu_i$$

This latest relationship between C and α_i allows us to impose a new constraint on α_i since $\mu_i \geq 0$, which is that $\alpha_i \leq C$.

This gives the new problem to solve in order to determine the weights of the support vectors:

$$\max_{\alpha} \sum_{i=1}^{L} \alpha_i - \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{H}\boldsymbol{\alpha}, \qquad 0 \leq \alpha_i \leq C \ \forall_i, \qquad \sum_{i=1}^{L} \alpha_i y_i = 0$$

The support vectors are then chosen for where the values of vector weights satisfy $0 < \alpha_i < C.$ b is calculated as before.

Section written in reference to [1] [2]

2.4 KKT Conditions

The standard KKT dual complementary condition apply for the SVM problem in dual form above. These can essentially be distilled into the following forms:

$$\alpha_i = 0 \ \Rightarrow y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

$$\alpha_i = C \ \Rightarrow y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \leq 1$$

$$0 < \alpha_i < C \Rightarrow y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$$

Which demonstrates how only for α's that are strictly within the bounds 0 and C do these correspond to the support vectors that define the placement of the hyperplane. Furthermore, this guarantees that an $\boldsymbol{\alpha}$ that satisfies these conditions will be an optimal solution of the SVM problem.

3 Further Topics in SVM

3.1 Kernels

In some instances, the two data sets are not separable by a hyperplane, and the pre-defined linear separation between the two sets is no longer applicable. The solution to this obstacle is to use a known function $\phi: \mathbb{R}^n \to \mathbb{R}^m$ to map the features of each point into a higher dimensional space in which the points *are* separable by a hyperplane, and to then apply the SVM algorithm to this new mapped set. This allows the SVM to continue to function even though the problem initially appears complex. This process is known as feature mapping.

In applying this technique, one will however quickly encounter an obstacle which is that applying feature mapping quickly increases the computational difficulty of the problem – consequently making the cost of applying this technique outweigh its benefit. Thankfully, one can circumnavigate this difficulty by applying the "kernel trick": that is replacing all of the inner products $\mathbf{x}_i \cdot \mathbf{x}_j$ with a function $K(\mathbf{x}_i, \mathbf{x}_j)$ which is defined as follows:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

In many instances, instead of calculating the values of $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$ directly, it is possible to reduce these directly to an expression involving only the inner product of $\mathbf{x}_i \cdot \mathbf{x}_j$ which is much simpler to calculate in terms of speed and memory.

Practically speaking, it is not necessary to know exactly what the expression for ϕ will be. According to the Mercer Theorem, K is a valid kernel if and only if the corresponding kernel matrix is symmetric positive definite.

There are a number of kernels frequently used with SVMs that are presented briefly here:

Linear Kernel

$$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$$

This is the most elementary of kernels that is simply the inner product of the two vectors. It is a benchmark for the minimum performance of a method. It is only suited for simple problems that can be linearly separated.

Polynomial Kernel

$$K(\mathbf{x}, \mathbf{y}) = (a + \mathbf{x}^T \mathbf{y})^b, \qquad a, b \in \mathbb{R}$$

This kernel is a popular choice in applications which allows one to map on to a space of dimension b, using a to control the weight of the higher order terms with respect to the lower order ones.

Sigmoid Kernel

$$K(\mathbf{x}, \mathbf{y}) = \tanh(a\mathbf{x}^T \mathbf{y} + b), \qquad a, b \in \mathbb{R}$$

Originally used in neural-networks this kernel often is benchmarked against other kernels presented here, but it appears to not be employed practically as often for SVMs.

Gaussian Kernel

$$K(\mathbf{x}, \mathbf{y}) = \exp(-a\|\mathbf{x} - \mathbf{y}\|), \qquad a \in \mathbb{R}$$

The Gaussian kernel is a popular choice for SVM machines and corresponds to a mapping feature of infinite dimensions.

Section written in reference to [3] [1]

3.2 More than Two Classes

An SVM by its nature can only differentiate between two classes, but in real-world scenarios datasets will require more than a simple binary classification. There are a number of methods for allowing an SVM to still be effective in these scenarios.

Common approaches involve decomposing a multiclass problem into a binary classification problem. The simplest is a "One-Versus-All" approach where each class is compared one-by-one to all the other classes. The means that for a problem with N classes, the SVM algorithm would need to be run N times. The weights for each data point are collected from each SVM run and the one with the highest value is considered to be the best match for the actual class.

Another option, which is more computational intense yet able to achieve better results in difficult scenarios is the "All-Versus-All" approach. In this setup, each class is compared one by one to each and every other class individually. For a problem with N classes, this leads to a total of $\frac{N(N-1)}{2}$ binary classifications. When processing a new point with an unknown class, it is passed through all the

SVMs and each claim whether the label of this point corresponds to the focus of the respective machine. The label with the greatest number of matches is then chosen as the best fit.

A third option, termed "Hierarchical Classification", arranges $N - 1$ classifiers to solve a problem with N classes. The approach is to create a sorting tree that accepts an unknown datapoint and progressively sorts it into different branches of the tree depending on the outcome of the binary classifiers at the nodes of each branch. This method is more challenging to implement as it requires careful thought into how the classifiers are arranged however it can achieve better performance than "All-Versus-All" strategies.

Section written in reference to [4]

4 THE SMO ALGORITHM

4.1 OUTLINE

Sequential Minimal Optimisation (SMO) is a technique that allows the resolution of an optimisation of a problem through the solution of many easier sub-problems. The approach is to optimise the problem repeatedly for a small subset of the independent variables, using different combinations of these in each iteration. The final solution can be checked to be optimal by ensuring it validates the KKT conditions.

In terms of accuracy and reliability the SMO algorithm is as good as any other solver available for addressing SVM training. In fact, its ability to work on small updates at a time can make it a preferable option when working with very large datasets that would pose difficulties to normal quadratic programs. The SMO is particularly well suited for SVMs with sparse inputs because of how it can easily exploit such features. Equally so, the SMO scales very well with the size of the dataset. This is because the basic operation at its core remains the same size regardless of the number of training points. Finally, SMO works well with kernels, meaning that even for problems with many attributes it can optimise the α's of the problem effectively.

The general outline of the algorithm is as follows:

```
Repeat until convergence:
    1. Select a pair of $\alpha_i$'s to optimise
    2. Optimise the pair of $\alpha_i$'s while holding all other $\alpha_i$'s constant
    3. Update the $b$ constant for the new set of $\alpha_i$'s
```

More detail about possible ways of implementing each step is presented in the following sections as well as which method was applied in the computational study for the last section.

Section written in reference to [5] [6] [1]

4.2 CHOOSING A PAIR OF $\boldsymbol{\alpha_i}$'S

The original method proposed by Platt in his first proposal of the SMO algorithm [5] repeatedly scans the dataset for α_i's that do not satisfy the KKT conditions and chooses one of these for optimisation along with a corresponding α_j. After passing over the set and optimising all the points once, the algorithm iterates a second time, however only selecting for optimisation the points that whose α_i's are neither 0 or C, and then optimising these with a corresponding α_j. Once this second iterating is complete, the process rebegins and continues to repeat until the KKT conditions are satisfied within a specified degree of tolerance.

In the above loop, once an α_i is chosen, the corresponding α_j is selected with the objective of maximising the size of the optimisation step. This is done by finding the α_j for which there is the *biggest* difference in the error of both α's which is given by the following:

$$|E_i - E_j|$$

Where:

$$E_k = \sum_{i=1}^{L} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x}_k + b$$

In some rare cases, the choice of α_i and α_j will not lead to an improvement in the final solution, at which point the SMO algorithm progressively widens its search for a new suitable α_j until one is found or if no α_j can be found then a new α_i is chosen.

4.3 Optimising the Pair of $\boldsymbol{\alpha_i}$'s

Once a pair of variables is chosen, they are optimised while holding all the others constant. This is the key strength of the SMO algorithm because this step can be done efficiently regardless of the size of the problem.

The first step of this part is to calculate the bounds on α_j. These are:

- $L = \max(0, \alpha_j - \alpha_i),\ \ H = \min(C, C + \alpha_j - \alpha_i)$ if $y_i \neq y_j$
- $L = \max(0, \alpha_i + \alpha_j - C),\ \ H = \min(X, \alpha_i + \alpha_j)$ if $y_i = y_j$

Second the new value for α_j is calculated with the following:

$$\alpha_j^{new} = \alpha_j + \frac{y_j(E_i - E_j)}{\eta}$$

Where $\eta = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)$

Then clipped to respect the bounds established in the first step:

$$\alpha_j^{new,clipped} = \begin{cases} H & \text{if} \quad \alpha_j^{new} \geq H \\ \alpha_j^{new} & \text{if} \quad L \leq \alpha_j^{new} \leq H \\ L & \text{if} \quad \alpha_j^{new} \leq L \end{cases}$$

Finally, the new value for α_i is found by

$$\alpha_i^{new} = \alpha_i + y_i y_j (\alpha_j - \alpha_j^{new})$$

4.4 Updating the $\boldsymbol{b}$ Constant

The b constant must also be updated as the optimisation progresses in order to keep it consistent in case either α_i or α_j are not at the bounds 0 or C. It's value is calculated such that the new values of α_i and α_j allow the SVM to still output the correct value of y_i for these updated weights. Thus, b is calculated as follows if α_i is not at the bounds:

$$b_1 = E_i + y_i(\alpha_i^{new} - \alpha_i)(K(\mathbf{x}_i, \mathbf{x}_i) + y_j(\alpha_2^{new,clipped} - \alpha_j)K(\mathbf{x}_j, \mathbf{x}_j) + b$$

Or it is calculated as follows if a_i is at the bounds, but α_j is not:

$$b_2 = E_j + y_i(\alpha_i^{new} - \alpha_i)(K(\mathbf{x}_i, \mathbf{x}_j) + y_j(\alpha_2^{new,clipped} - \alpha_j)K(\mathbf{x}_j, \mathbf{x}_j) + b$$

If both α's are not at the bounds then the value for b is given by

$$b = \frac{b_1 + b_2}{2}$$

Section written in reference to [2] [5] [6]

5 Simulation Study

5.1 Overview

In this section, the SMO algorithm as described above is applied to resolve an SVM for the classification of the *Iris* dataset. A small study is performed where two different kernels (polynomial and gaussian) are applied to this problem, and we observe how this impacts the speed and accuracy of the SVM. The SVM is written in Python and uses the NumPy library.

5.2 The IRIS Dataset

The Iris dataset is a well-known set of measurements of flowers used often in machine learning examples. It consists of 150 data points: each with four attributes (*sepal length, sepal width, petal length, petal width*) and an associated class corresponding to one of three flower species (*setosa, versicolor, virginica*). The challenge presented by this dataset is that while *setosa* species is linearly separable from the other two labels, these latter ones are not linearly separable from each other. [7]

5.3 Performance Comparisons

5.3.1 Method

The two kernels that are compared here are the polynomial and the gaussian kernels. These were presented in the section earlier. The constants for the polynomial kernel are set to $a = 0$ and $b = 2$, and for the gaussian kernel, the constant was $a = 2$. These constants were set empirically as they appeared to give the best performance.

The algorithm that is implemented is a simplified version of the SMO algorithm described above but uses a different heuristic for choosing which α's to implement – this version of the algorithm was presented by [6].

In this particular case a_i is selected as normally in the SMO procedure, but α_j is the chosen at random out of the remaining weights and it is evaluated for whether it can be optimised. If not then the algorithm chooses another α_i and makes another random choice of α_j. The algorithm continues to iterate over α_i's trying different α_j's, and it will iterate over the whole range of α's several times (a parameter passed to the algorithm) until supposing that all the pairs have been optimised fully and terminating. The complete maximum number of iterations is also limited to a user-set parameter.

In this algorithm, after a successful update to α_i and α_j, this new version of $\boldsymbol{\alpha}$ is saved to memory. Once the algorithm finishes, the collection of $\boldsymbol{\alpha}$'s is used to plot the convergence of the algorithm: the distance from the "perfect solution" found by a commercial Quadratic Program solver is plotted for each update. This allows for the comparison between different kernels being applied to the SMO. A simple "One-Versus-All" classifier is then implemented to assess how well the SVM works for multiclass classifications.

As a further metric of performance, we also assess the SVM's performance in binary classification by measuring the time needed for training and in its accuracy of classifying new points. The dataset is split into a training and a testing set with at 80/20 random split of the original dataset, and the measurements are repeated several times with different random allocations of training/test data.

5.4 Results

From the test results it is immediately obvious that the applying the Gaussian kernel leads to more accurate and to quicker convergence results for this particular problem. The solution employing the Guassian kernel gets close to 100% accuracy during testing and can converge much quicker; whereas the result achieved when using the polynomial kernel, is below 70% and much less reliable in how quickly it converges. These test results are presented in the first table. This also puts into context the multiclass sorting results. Again, the solution using the Gaussian technique is much more accurate than the polynomial – probably because its ability to determine between different classes is also much more accurate.

The reason for the poor performance of the polynomial kernel appears to be due to an instability in the algorithm for remaining close to the ideal solution. There is an indication of this in the two plots of the polynomial technique's convergence: it can be seen that as it approaches a solution close to the ideal, there is much more variation in its approach to the solution, and in one case diverges from it towards the end of its computation.

The general behaviour of the SMO algorithm agrees with the reported and theoretical behaviour that it can converge quickly; however the fact that the polynomial kernel leads to poor performance may be a consequence of implementing this specific SMO algorithm.

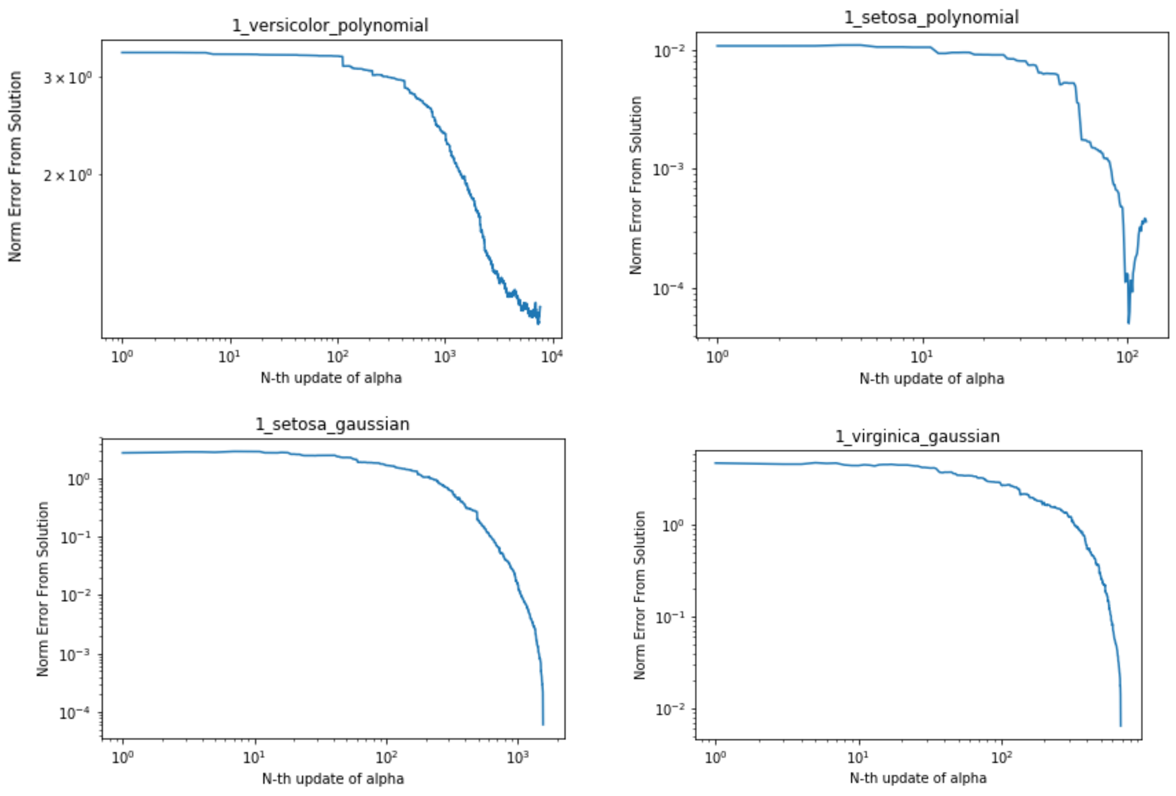
Evaluation 1: Binary Classification – Average of five tests

	Polynomial			Gaussian		
Class of Interest	**Accuracy (%)**	**SMO Iterations**	**Training Time (s)**	**Accuracy (%)**	**SMO Iterations**	**Training Time (s)**
Versicolor	67	10144	265	98	923	76
Setosa	65	60	19	100	1099	99
Virginica	69	1143	175	95	751	66

Evaluation 2: Multiclass Classification – Average of five tests

	Polynomial	**Gaussian**
Accuracy	44%	98%

Figure 3 – Convergence plots for the SVM training



6 CONCLUSION

This study briefly reviewed the theory behind SVMs, kernels and the SMO algorithm. In its proposed study, it applied a SMO solution for creating and SVM that could classify the Iris dataset, and it compared the performance of the gaussian kernel and the polynomial kernel against each other. The results of this study indicate that the gaussian kernel is a much more resilient solution for the SMO implemented in this work when applied to the Iris dataset.

7 REFERENCES

[1] Andrew Ng, *CS229 Lecture Notes,* Stanford University.

[2] T. Fletcher, *Support Vector Machines Explained,* University College London, December 2008.

[3] C. Souza, “Kernel Functions for Machine Learning Applications,” crzousa.com, 17 March 2010. [Online]. Available: http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/ . [Accessed 19 May 2019].

[4] M. Aly, “Survey on Multiclass Classification Methods,” Caltech, November 2005.

[5] J. C. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines,” Microsoft Research, 1998.

[6] Stanford University, *CS 229, Autumn 2009: The Simplified SMO Algorithm,* 2009.

[7] UCI Machine Learning Repository, “Iris Data Set,” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/iris.