Communications and Information Engineering Program
Linear and Nonlinear Programming
MATH 404 - Fall 2020

Project Report

Sequential Minimal Optimization for Support Vector Machine Binary Classification

Prepared by
ElSayed Mostafa 201700316
Salma Elbess 201601152

Table of contents

Abstract

Support Vector Machines (SVM) is one of the most popular supervised machine learning models recently. SVM models are used extensively wherever in classification and regression problems. Its objective is to find a hyperplane H_1 that separates the classes/clusters of the dataset. Hence, an optimization problem is created to minimize the cost function of the SVM model to obtain acceptable accuracy. The main parameters to apply the optimization upon in SVM are the parameters of two other hyperplanes H_2 and H_3 for which the margins around H_1 is maximized or ,generally speaking, achieving better performance. In this project, SVM optimization problem is solved using the sequential minimal optimization algorithm (SMO) with the help of dual-primal problem and Lagrangian multipliers. The SMO algorithm for SVM is also implemented using MATLAB and compared with the ready-made MATLAB function to solve SVM using SMO in addition to comparison of performance with a benchmark problem.

Introduction

In recent years, data science has been given a lot of awareness and support due to its daily used applications. Machine Learning is a subfield of data science that merges data science and statistics is a way that builds predictive models for a specific phenomenon to be used in smart applications. One of the main problems in machine learning is classification; classification means building a model that correctly assigns the input data to a specific class based on the manipulation and the processing of historical training data. A lot of classification algorithms are extensively used in machine learning such as Logistic Regression, K-Nearest Neighbours and decision trees. In any classification algorithm first development, mathematical optimization plays the master rule in such development. Using the optimization algorithms and solver generates the solution to the classification problem in different forms that generally enable the machine learning developers using these forms easily to train the model with the data and then obtain the optimal model for classification.

Recently, support vector machines SVMs are highly trusted as one of the strongest machine learning classification models. In the field of machine learning and data analysis in general, it is often witnessed that SVM performance is used extensively more than the other classification algorithms. The base idea for SVM is to generate an optimum hyperplane that separates the points in two different regions on the sides of the hyperplane. For targeting this high performance, different manipulation and modification steps have been proposed and studied for SVMs. The most popular basic approach is using slack variables to allow training points existence with the margin between the support vectors. Therefore, the hard-margin SVM problem is modified to be as the soft-margin SVM problem with these slack variables. Moreover, SVM is used for both binary and multiple classification problems such as in one-against all SVM algorithm.

Due to the nature of SVM classifiers, it is originally suitable for the linearly separable data. To use SVM in non-linearly separable data, a technique called "***kernel trick***" is used. The idea of kernel trick is to replace the original data features, which are not linearly separable, by another set of hypothetical features that is more linearly separable and also representative of the data. Hence, different similarity functions are used to generate this type of features and replace the dot product between any two point vectors such as polynomial kernel, Gaussian kernel and Laplace LBF kernel. By using any kernel function, the SVM usage is more efficient in different data distributions. Different mathematical optimization techniques are used to get the optimal hyperplane for SVM problems.

As a quadratic optimization problem, the SVM problem is solved using different quadratic optimization solvers such as interior point methods and gradient projection method [1]. Within the popular used optimization algorithms in SVM, the sequential minimal optimization (SMO) algorithm is used frequently in literature due to its edge in time consumption. SMO algorithm divides the optimization problem into sub optimization problems and solves them analytically without the numerical QP solvers. To have more variables to optimize and hence obtain much more trusted results, the training data ,in case of SVM, is used instead of the original data features for the optimization problem. In this project, a clear description about the SMO algorithm, its procedure and implementation is given and tested for binary classification problems using MATLAB.

Problem formulation

For binary SVM classifier, a hyperplane $f(x)$ is used to separate the two classes. The formula of this hyperplane is:

$$f(x) = w^T x - b$$

where the output of the classifier is determined depending on the location of the point to the hyperplane using the relations:

$$y_i = 1 \quad at \quad w^T x_i - b > 0$$
$$y_i = -1 \quad at \quad w^T x_i - b < 0$$

As the idea of SVM is to use the nearest points from the optimal separating hyperplane to form a marginal area in order to maximize this marginal area and hence get the most separation between the two classes, other hyperplanes are proposed parallel to the separating one and at equal distances from both sides (as shown in figure 1). Hence, the inequalities to the optimization problem can depend on these hyperplanes to represent the needed marginal area.

$$y_i = 1 \quad at \quad w^T x_i - b \geq 1$$
$$y_i = -1 \quad at \quad w^T x_i - b \leq -1$$

The former two inequalities can be merged in one inequality.

$$y_i(w^T x_i - b) \geq 1$$

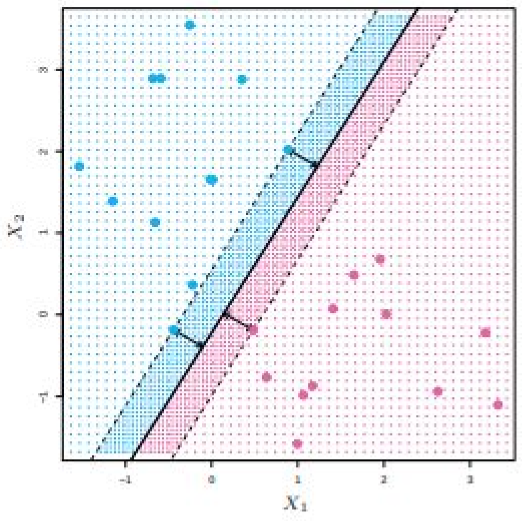

Figure (1): SVM problem and margins at the support vectors **[2]**

Suppose a point x on one of the marginal hyperplanes, its euclidean distance to the main hyperplane is simply $\frac{|f(x)|}{\|w\|}$ which in turns equal $\frac{1}{\|w\|}$. Therefore, the distance between the two marginal hyperplanes, which is needed to be maximum, is $\frac{2}{\|w\|}$ or equivalently $\frac{2}{w^T w}$. Hence, the primal optimization function is now fully formulated:

$$\textbf{max} \ \frac{2}{w^T w}$$
$$\textbf{subject to} \ y_i(w^T x_i - b) \geq 1 \ , \ i = 1, 2, ..., N$$

A better formula can be got by replacing the maximization with minimization problem:

$$\textbf{min} \ \tfrac{1}{2}w^T w$$
$$\textbf{subject to} \ y_i(w^T x_i - b) \geq 1 \ \ , \ i = 1, 2, \ldots.., N$$

To solve the former problem, the constraint problem is converted into unconstrained one using the lagrangian multipliers as:

$$\textbf{min} \ Q(w, b, \alpha) \equiv \tfrac{1}{2}w^T w - \sum_{i=1}^{N} \alpha_i(y_i(w^T x_i - b) - 1)$$

where α_i are the nonnegative lagrangian multipliers, $i = 1, 2, \ldots.., N$.
The optimal solution for this convex quadratic programming problem is achieved at the point satisfies the KKT conditions [3]:

$$\frac{\partial Q(w,b,\alpha)}{\partial w} = 0$$
$$\frac{\partial Q(w,b,\alpha)}{\partial b} = 0$$
$$\alpha_i(y_i(w^T x_i - b) - 1) = 0 \ \text{for} \ i = 1, 2, \ldots.., N$$
$$\alpha_i \geq 0$$

By solving the two previous partial differentiations, the two formulas is extracted respectively:

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

By substituting these last two formulas in the optimization problem, we get the dual problem

$$\textbf{max} \ Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$$
$$\textbf{subject to} \ \sum_{i=1}^{N} \alpha_i y_i = 0 \ \ \textbf{for} \ \ i = 1, 2, \ldots.., N$$

As it is a concave dual quadratic programming problem with a convex primal quadratic programming problem, the zero gap is maintained between the two problems hence, solving the dual problem is exactly the same as solving the primal one [3].

After solving the former problem (solving for the values of α_i), the values of w can be obtained from the relation ($w = \sum_{i=1}^{N} \alpha_i y_i x_i$). Also, the threshold values (b) can easily obtained through the main relation by: $b = w^T x_i - y_i$ for each point (i) and then getting the average value.

All the previous analysis is for the hard-margin SVM problem, where no tolerance is given for any point to lie inside the margin even if it was correctly classified. Therefore, to smooth the optimization problem and to be less sensitive to each individual point, a slack variable is introduced to each point to enable its existence within the margin as long as it's correctly classified (as shown in figure 2).

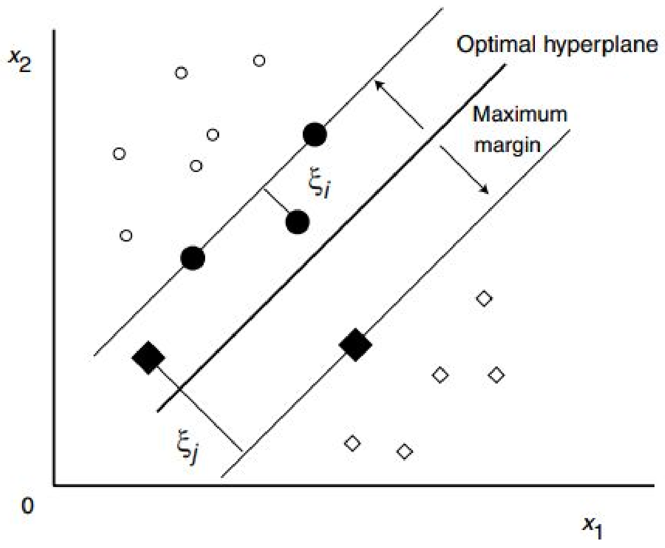

Figure (2): Slack variables added to the SVM problem **[3]**

In this case, the slack variables ξ are added to the inequality constraints.

$$y_i(w^T x_i - b) \geq 1 - \xi_i \quad , \; i = 1, 2, ..., N$$

To obtain the best (widest) margin with correct classification, the optimization primal problem is modified to include the values of ξ. The new primal problem is

$$\textbf{min } Q(w) = \tfrac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

$$\textbf{subject to } y_i(w^T x_i - b) \geq 1 - \xi_i \quad , \; i = 1, 2, ..., N$$

where C is a hyper-parameter controls how tolerant the optimal hyperplane is towards having points inside the marginal area; higher C means less tolerance and hence wider margin. The same analysis is applied to this soft-margin problem as done for the hard-margin one. The resultant dual problem obtained is the same as the hard-margin problem with the same expression of w, b and the same constraints except that $0 \leq \alpha_i \leq C$ instead. The final problem and constraints formulation is as the following.

$$\textbf{max } Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

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

$$0 \leq \alpha_i \leq C \textbf{ for } \; i = 1, 2, ..., N$$

This problem is then solved using the sequential minimal optimization (SMO) algorithm.

SMO algorithm

As an optimization algorithm, the sequential minimal optimization algorithm mainly targets partitioning the problem in hand to several subproblems that can be easily solved. It consists of two main steps, first the algorithm applies heuristics to choose two lagrangian multipliers to optimize. Once the two multipliers are chosen, simple algebra and calculus can solve for the optimum value for each α. The main advantage of the SMO algorithm is that the time taken for a solution is linear in the size of the training set which is a way better than other algorithms where their time may be linear or quadratic in the training set size [4].

Choosing Two multipliers for optimization

There should be at least one multiplier that violates the KKT conditions in the optimization subset in each iteration. This theorem is proved by Osuna [5] . So, at least the first multiplier is tested against the KKT conditions to ensure that they are violated. The SMO algorithm proposed in Platt 1998 [4] uses some heuristics in choosing the first and second multiplier to speed up the process.
To choose the first multiplier, the algorithm iterates over the samples to find one that violates the KKT conditions within some accepted tolerance (proposed in Platt to be 0.001). First, the algorithm iterates over all the samples. Then, It iterates over the non bound examples (example with alpha not equals to zero nor C) as the bound examples are less expected to violate the KKT conditions according to Platt 1998[4]. So, it is more time efficient to iterate over examples more suspected to violate the KKT condition first and make them consistent with each other then check the other examples. After all the non-bound are optimized, the algorithm iterates over all the samples , bound and unbound. The algorithm alternates between checking the entire dataset and the non-bound examples until there is no change in weights within some tolerance and the algorithm terminates.
After choosing the first multiplier, the algorithm has three heuristics to choose the second one. The algorithm starts checking them in the following mentioned order and if one fails the next is used. According to Platto[4] , the first heuristic to choose a non- bound lagrange multiplier that maximizes the difference in alpha. As alpha follows the equation $\alpha_2 = \alpha_2^{old} - \frac{y_2(E_1 - E_2)}{\eta}$ (discussed in detail in the next section), the algorithm chooses the multiplier that maximizes the absolute difference between E_1 and E_2.
If the algorithm fails to optimize the two multipliers together, it starts iterating over all the non bound examples starting from a random position. While iterating, the algorithm tries to optimize the two multipliers. If the algorithm fails to find any second multiplier that can be optimized with the first chosen one within the non-bound subset, then it examines the entire dataset.
The algorithm iterates over the entire examples starting from the random location trying to find the joint optimization of any multiplier with the first multiplier.
The random start in the second and the third heuristics ensures that the SMO algorithm is not biased towards the examples as the beginning.

Joint optimization

Once selecting two lagrangian multipliers $(\alpha_1,\ \alpha_2)$, the SMO optimizes them jointly. Looking at the situation, it is found that the multipliers are constrained by to constraints which are:

$$0 \le \alpha_1, \alpha_2 \le C$$
$$\alpha_1 + \alpha_2 = k \quad if\ \ y_1 = y_2$$
$$\alpha_2 - \alpha_1 = k \quad if\ \ y_1 \neq y_2$$

where k is the summation of the remaining multipliers based on $\sum_{i=1}^{N} \alpha_i y_i = 0$ which is constant. Therefore, the values of α is represented within a line inside a square (as shown in figure 3). To keep this condition valid all the time, $(\Delta\alpha_1 =- y_1 y_2 \Delta\alpha_2)$ will be kept for each joint optimization.

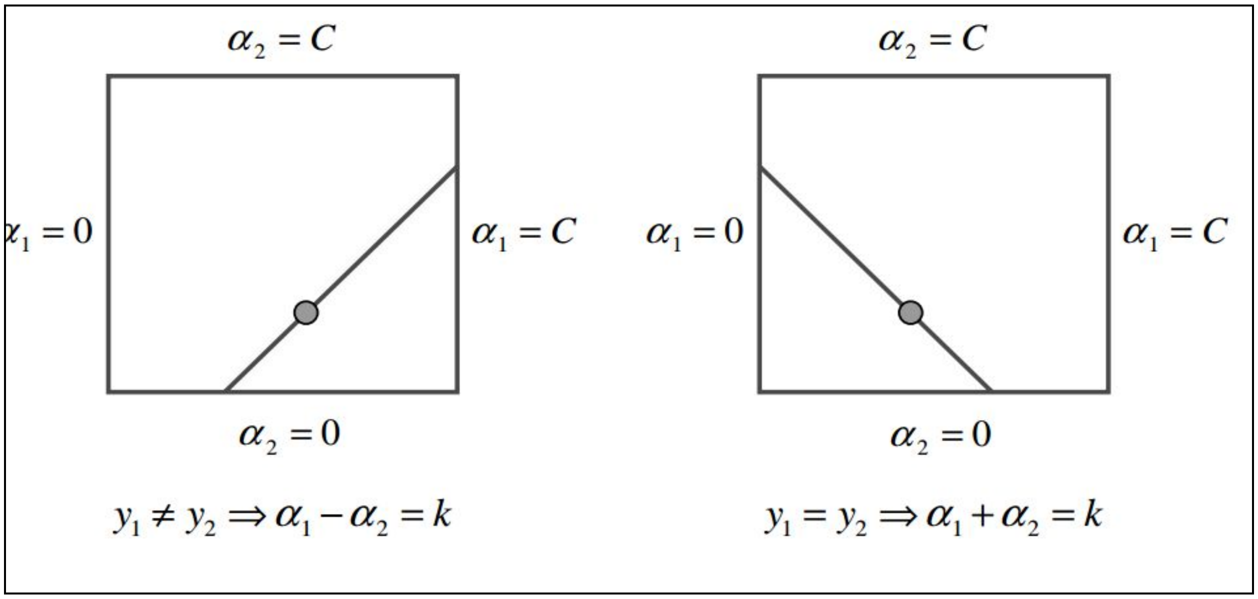

Figure (3): The constraints on any two lagrangian multipliers **[4]**

The problem of this joint optimization is constrained minimization problem. The first step is to calculate the constraints (limits) on one of the multipliers to be within the diagonal line (as shown in figure 3). To get the appropriate limits for α_2, the two multipliers must be kept valid. In case of $y_1 = y_2$ and $k > C$, the value of α_2 can't be less than their difference $(k - C)$ in order to make sure that α_1 will be less than or equal C even with the lowest possible values of α_2 then which is $(k - C)$. Another case is when $y_1 = y_2$ and $k < C$, the value of α_2 can't exceed k in order to make sure that α_1 will be nonnegative even with the highest possible values of α_2 then which is k. All the conditions for these constraints to get the lowest value L and the highest value H are summarized as follows:

- When $y_1 = y_2$:
  $L\ =\ max(0, \alpha_2 + \alpha_1 - C)$ $\qquad$ $H\ =\ min(C, \alpha_2 + \alpha_1)$
- When $y_1 \neq y_2$:
  $L\ =\ max(0, \alpha_2 - \alpha_1)$ $\qquad$ $H\ =\ min(C, C + \alpha_2 - \alpha_1)$

Next, the objective function $Q(\alpha)$ is solved by substitution of α_2, α_1 while keeping the other multipliers constant.

$$D \equiv Q(\alpha_1, \alpha_2) \ = \ \alpha_1 + \alpha_2 - \tfrac{1}{2}(\alpha_1^2 y_1^2 x_1^T x_1 \ + \alpha_2^2 y_2^2 x_2^T x_2 + 2\alpha_1 y_1 \alpha_2 y_2 x_1^T x_2 \ + \ 2(\alpha_1 y_1 x_1^T + \alpha_2 y_2 x_2^T)(\sum_{i=3}^{N} \alpha_i y_i x_i^T \ ) \ + R\ )$$

where $R \ = \sum_{i=3}^{N} \alpha_i + \ \sum_{i=3}^{N}\sum_{j=3}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j$ is a constant term.

To make the equation more readable, $x_i^T x_j$ is replaced by K_{ij} and $s = y_1 y_2$

So, $D = \ \alpha_1 + \alpha_2 - \tfrac{1}{2}(\alpha_1^2 K_{11} \ + \alpha_2^2 K_{22} + 2\, s\, \alpha_1 \alpha_2 K_{12} \ + \ 2(\alpha_1 y_1 x_1^T + \alpha_2 y_2 x_2^T)(\sum_{i=3}^{N} \alpha_i y_i x_i^T \ ) \ + R\ )$

To make a function in α_2 only, the old value of $\alpha_2 + s\alpha_1$ is used (k). Hence, $\alpha_1 = k - s\alpha_2$.
By applying the former substitution of α_1 with some straightforward algebra, it is easily shown that

$$D = \tfrac{1}{2}\alpha_2^2(2K_{12} - K_{11} - K_{22}) \ + \alpha_2\ (1 - s + sK_{11}k - sK_{12}k + y_2 \sum_{i=3}^{N} \alpha_i y_i x_i^T x_1 \ - y_2 \sum_{i=3}^{N} \alpha_i y_i x_i^T x_2\ ) + CONST$$

Further simplifications for D with replacing k by $\alpha_1^{old} + s\ \alpha_2^{old}$ and introducing the error terms E_1 and E_2, the formula of D becomes

$$D = \tfrac{1}{2}\alpha_2^2(2K_{12} - K_{11} - K_{22}) \ + \alpha_2(\ y_2(E_1 - E_2) - (2K_{12} - K_{11} - K_{22})\ \alpha_2^{old}) \ + \ CONST$$

The equation D is now ready to be differentiated w.r.t. α_2 and then maximized by letting the first derivative $D_1 = 0$ and with negative value for the second derivative η.

$$D_1 \equiv \ \eta\alpha_2 + y_2(E_1 - E_2) - (2K_{12} - K_{11} - K_{22})\ \alpha_2^{old} = 0$$

Then $\alpha_2 \ = \ \alpha_2^{old} - \frac{y_2(E_1 - E_2)}{\eta} \rightarrow \mathbf{(1)}$

where $\eta = 2K_{12} - K_{11} - K_{22}$

Using the update rule in (1), the new value of α_2 is set calculated but before using it as the final value of α_2, it must be clipped first between L and H if it exceeds these two limits.
Next, the new value of α_1 can be calculated using $\Delta\alpha_1 =- s\Delta\alpha_2$. So $\alpha_1^{new} = \alpha_1^{old} - s(\alpha_2^{new} - \alpha_2^{old})$.
In case of non-negative value of η which does not guarantee maximizing the objective function, the value of α_2 is set to the value of its bounds which maximize the objective function more otherwise, the value of α_2 is not modified within such an iteration.

After finishing any iteration (updating 2 lagrangian multipliers), the values of the original weights vector is updated using the previously proved rule $w \ = \ \sum_{i=1}^{N} \alpha_i y_i x_i$ in order to update the errors calculated in each iteration. Also, the threshold value is updated using the main rule that: $b \ = \ w^T x_i \ - \ y_i$ for each example and averaging the result; however, it is very smart to calculate the values of the non-optimized example. Instead, the difference originating from the two optimized points is used in addition to the error originated from the new weight vector usage. Hence, depending on which α_i lies inside its bounds ($L,\ H$), the value of new threshold b is calculated from $b \ = \ b^{old} + E_i + y_1(\alpha_1^{new} - \alpha_1^{old})x_i^T x_1 + y_2(\alpha_2^{new} - \alpha_2^{old})x_i^T x_2$

Methodology and Implementation

Based on the discussed algorithm in SMO algorithm section, MATLAB scripts are written to implement this algorithm. More hyperparameters are added to compensate for the difficulty for having exact zero in the KKT conditions for example. The implementation is generally consists of two main function; ***'examinExample'*** function to choose the lagrangian multipliers to optimize ant ***'take_step_smo'*** function to apply the joint optimization for the two chosen multipliers and then updates the weight vector w and the threshold b. The ***'examinExample'*** function contains the three different approaches of choosing the multipliers in three different MATLAB files. For training, testing and comparing the model, two MATLAB scripts are written to get the data, set the parameters, fit the model and test. The first script is named ***'mainfile_artificial_data'*** in which artificial data is generated and then the model is applied to it. Using the actual classes values and the predicted ones, the training error is calculated. Similarly, a script named ***'mainfile_real_data'*** is used to use a ready real linearly separable dataset in the classification SVM. The results in this script are compared to the results obtained using the same dataset by the ready-made MATLAB SVM classifier using the SMO algorithm. The scripts are all available at the following link.
https://drive.google.com/file/d/1TQMjGYXu7Om3HCywZGUkbIubjZs7J05S/view?usp=sharing

Results and Discussion

In order to test our MATLAB implementation with the one generated by J. Platt using C++ programming language and the other MATLAB built-in implementation [4]. Two different datasets are used to train and test the implementation of SMO algorithm. As generated by J. Platt, a random input data is generated in a way that suits the linear kernel used in our implementation.

Results

Artificial Dataset

To generate the random artificial data, each data example is assumed to be a 300-dimensional binary vector in which the ratio of ones is 10% of the vector components. To generate the target vector of -1 and 1 that inhibit some relation between each input example and its target, a random vector of [-1,1] is generated and then multiplied by each vector of the input data by dot product operation. Depending on the value of the dot product, the target value is either 1 or -1 if it is greater than 1 or less than -1 respectively; otherwise, the example point is discarded [4]. Once the artificial data is generated, our SMO model is fitted and its results of time consumption and training accuracy is calculated and compared with the reference output time as in table 1. For this comparison the value of C is 100.

Sample Size (in our implementation)	**Our Implementation Time** (seconds)	**Training Accuracy**	**Time in [4]** (seconds)
1000 (1019)	70.589	100 %	15.3
2000 (2061)	153.75	99.9%	33.4
5000 (5079)	368.048	100 %	103.0

Table (1). Comparison for artificial data output using SMO algorithm for SVM **[4]**

Bills Authentication Dataset

For further investigation to our model's accuracy and comparison with the other ready-made functions. A dataset from UCI repository about bills authentication is used to be fitted and tested using our MATLAB implementation and the built-in SVM solver in MATLAB. The dataset consists of 4 different features and binary target values. The number of instances in the dataset is 1372 [6]. The parameters are set to be C = 100, KKT conditions Tolerance = 0.001. The measured time is in seconds.

Sample Size	Our Implementation Training Time	Matlab built-in function Training Time	Our Implementation Training accuracy	Our Implementation Test accuracy	Matlab built-in function Test accuracy
200	0.212	0.017	100%	100%	100%
400	0.340	0.021	100%	97.5%	97.5%
500	6.939	0.128	99.11%	98%	98%
600	5.565	0.37	99.07%	98.33%	96.67%

Table (2). Comparison between Matlab built-in function and our implementation using Bills Authentication Dataset

The results verified in tests of the same sample size due to data shuffling. However, these results are the observed mean.

Weights and bias results for a one sample size = 200 are as follows
Matlab weights = [-2.6905, -0.9682, -1.4606, 0.1779]
Our implementation weights = [-2.6221, -0.9711, -1.4531, 0.1832]
Matlab bias (threshold) = 2.4253
Our bias (threshold)   = -2.3799

Discussion

Artificial Dataset

The results in table 1 reveals that our MATLAB implementation is consistent with the advantage of SMO algorithm which states that the execution time is linearly proportional to the sample size. As a comparison with the results obtained by J. Platt [4], the same trend of relation between the sample size and execution time is easily revealed. However, the difference in time values is due to the difference between MATLAB and C++ execution speeds; it is known that C++ is extremely faster than MATLAB [7]. For the slight differences in the sample size, it is due to the way our random data is generated is different from the way it is generated by J. Platt; however, these few more data examples are assumed to have no obvious effect on the results. The target from this comparison is already achieved due to the exact correlation between the execution time and the sample size besides the accurate model results obtained on the training data.

Bills Authentication Dataset

Matlab built-in function is faster than the proposed implementation. However, the accuracy of our implementation is comparable to the accuracy of the built-in function, sometimes our was higher. All the accuracies are higher than 90% for the tested sample sizes. The weights and bias are almost the same in both the implemented and built-in functions in most of the cases. However when they differ, the accuracy of the produced svm classifier is still within the same range. The difference in the bias sign due to the different equation used for svm. In our implementation we used it as -b in the svm equation but the built-in function uses +b in the svm equation.

Conclusion

Using the SMO algorithm, the SVO problem is solved for binary classification. SMO is an optimization algorithm that breaks the original quadratic optimization problem to sub optimization problems and then solves them analytically. The SMO algorithm consists of two main steps which are choosing the two lagrangian multipliers to optimize and then apply optimization to the chosen multipliers analytically. With additional hyperparameters for tolerance, the SMO algorithm for binary classifier SVM is implemented in MATLAB. Different dataset is used to train the model and compare the implementation performance based on its speed and its accuracy. The results reveal the good accuracy of the implemented algorithm in addition to the suitable CPU time.

References

[1] J. Nocedal and S. Wright, *Numerical optimization*, 2nd ed. New York, NY: Springer, 2006, pp. 480:490.
[2] G. James, D. Witten, T. Hastie and R. Tibshirani, *An introduction to statistical learning with applications in R*. 2013, pp. 341:358.
[3] S. Abe, *Support Vector Machines for Pattern Classification*, 2nd ed. New York: Springer, pp. 21:60.
[4] Platt, John. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Advances in Kernel Methods-Support Vector Learning. 208.
[5] Osuna, E., Freund, R., Girosi, F .(1997) . Improved Training Algorithm for Support Vector Machines, Proc. IEEE NNSP '97.
[6]V. Lohweg and H. Doerksen (2013). UCI Machine Learning Repository [https://archive.ics.uci.edu/ml/datasets/banknote+authentication#]. Ostwestfalen-Lippe, University of Applied Sciences.
[7] Andrews, Tyler. (2012). Computation Time Comparison Between Matlab and C++ Using Launch Windows.