INTERNATIONAL JOURNAL OF COMPUTATIONAL MATHEMATICS AND DATA SCIENCE

IMPLEMENTATION OF SUPPORT VECTOR MACHINE LEARNING ALGORITHM USING STOCHASTIC SUB-GRADIENT DESCENT

[1]Kalu Stephen I. (PhD), [2]Ugorji Clinton Chikezie (PhD)

[1]School of Science & Industrial Technology, Department of Statistics, Ogbonnaya Onu Polytechnic, Aba.
[2]School of Science & Industrial Technology, Department of Computer Science, Ogbonnaya Onu Polytechnics, Aba.

Correspondence Author Email: [1]monarchynwokike@gmail.com

DOI: ***https://doi.org/10.5281/zenodo.17608177***

Abstract

Support Vector Machines (SVMs) are among the most robust supervised machine learning algorithms for classification and regression tasks. They aim to find the optimal hyperplane that maximizes the margin between classes in a dataset. However, traditional quadratic programming-based optimization of SVMs can be computationally intensive for large-scale problems. This paper provides a detailed, step-by-step implementation of the SVM learning algorithm using the Stochastic Sub-Gradient Descent (SSGD) optimization technique. The study discusses the mathematical formulation, algorithmic structure, convergence properties, and computational advantages of SSGD-based SVMs. Empirical results and pseudo-code illustrations demonstrate the efficiency and scalability of the approach.

Keywords: Support Vector Machine, Stochastic Sub-gradient Descent, Optimization, Machine Learning, Classification.

1. Introduction

Machine learning has evolved as a cornerstone of artificial intelligence, enabling systems to learn patterns and make decisions without explicit programming. Among the numerous algorithms developed, the **Support Vector Machine (SVM)** stands out for its strong theoretical foundation in statistical learning theory (Vapnik, 1995). SVMs work by identifying an optimal hyper-plane that separates data points of different classes with the maximum margin.

Despite their high accuracy and generalization ability, traditional SVMs suffer from scalability issues when applied to large datasets. The quadratic optimization problem inherent in SVM formulation becomes computationally prohibitive as the number of samples increases. Consequently, researchers have sought efficient optimization techniques, one of which is **Stochastic Sub-Gradient Descent (SSGD)** — an iterative optimization method suitable for large-scale and streaming data environments.

This paper provides a structured, step-by-step exposition of how the SVM learning algorithm can be implemented using SSGD. The goal is to combine the robustness of SVMs with the scalability of stochastic optimization.

2. Literature Review

Support Vector Machines were first introduced by **Cortes and Vapnik (1995)** and have since been applied extensively in pattern recognition, text classification, bioinformatics, and financial modeling. Early SVM implementations relied on quadratic programming solvers (Platt, 1998), which are unsuitable for big data applications.

Bottou (2010) proposed stochastic gradient methods as an efficient way to train large-scale models by updating parameters based on one or few samples per iteration. Later, **Shalev-Shwartz et al. (2011)** demonstrated the efficiency of **Pegasos (Primal Estimated sub-Gradient Solver for SVM)**, which applies stochastic sub-gradient descent directly to the primal SVM objective function.

Further advancements by **Hsieh et al. (2008)** and **Chang & Lin (2011)** emphasized the need for scalable algorithms for linear SVMs in high-dimensional feature spaces. In modern deep learning contexts, variants of stochastic optimization such as Adam, RMSProp, and Adagrad have evolved, yet the simplicity and robustness of SSGD make it a preferred choice for classical SVM training.

3. Materials and Method of SVM

3.1 Primal Formulation

Given a dataset of n samples (x_i, y_i), where $x_i \in \square^d$ and $y_i \in -1, +1$, the linear SVM aims to solve:

$$\min_{w,b} \tfrac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w.x_i + b)) \qquad (1)$$

Where w is the vector weight, b is the bias, C is the regularization parameter, and the term $\max(0, 1 - y_i (w.x_i + b))$ represent the hinge loss.

3.2 Objective Function

DOI: https://doi.org/10.5281/zenodo.17608177

The primal objective can be expressed as:

$$L(w) = \frac{1}{2} \| w \|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w.x_i + b)) \quad (2)$$

Our goal is to minimize $L(w)$ using **stochastic sub-gradient descent**.

3.3 Stochastic Sub-Gradient Descent (SSGD) Method

3.3.1 Concept

SSGD is an iterative optimization method that updates the model parameters using the sub-gradient of the loss function computed from a randomly selected subset (mini-batch) of training data.

4.2 Sub-gradient of the Hinge Loss

For each sample (x_i, y_i):

$$\nabla_w = \begin{cases} w - C y_i x_i, & if\ y_i (w.x) < 1 \\ w, & otherwise \end{cases} \quad (3)$$

3.3.2 Update Rule

At iteration t, with learning rate η_t:

$$w_{t-1} = w_t - \eta_t \nabla_w \quad (4)$$

where $\eta_t = \frac{1}{\lambda t}$, and λ is the regularization coefficient controlling convergence.

3.3.3. Step-by-Step Implementation

Step 1: Initialize Parameters

- Initialize $w = 0, b = 0$
- Choose η_0 (initial learning rate) and C.

Step 2: Shuffle Dataset

- Randomly shuffle training samples to reduce correlation.

Step 3: Iterate Over Epochs

For each epoch:

- Select a random subset (mini-batch) of training samples.
- For each sample (x_i, y_i), compute :
  - The margin
  - The sub-gradient ∇_w
- Update:

$$w = w - \eta_t \nabla_w \quad (5)$$

- Optionally update bias:

$$b = b + \eta_t C y_i \ \ if \ \ y_i(w.x_i + b) < 1 \quad (6)$$

Step 4: Decay Learning Rate

Reduce η_t gradually for stable convergence.

Step 5: Evaluate Convergence

Continue iterations until changes in the loss function are minimal or a fixed number of epochs are completed.

4.0 Data Presentation and Analysis

A data problem was devised so that the classes are linearly separable. This means that a straight line can be drawn to separate the classes. This is intentional so that we can explore how to implement an SVM with a linear kernel (straight line). An assumption made by the SVM algorithm is that first class value is -1 and the second class value is +1.

X1	X2	Y
2.327868056	2.458016525	-1
3.032830419	3.170770366	-1
4.485465382	3.696728111	-1
3.684815246	3.846846973	-1
2.283558563	1.853215997	-1
7.807521179	3.290132136	1
6.132998136	2.140563087	1
7.514829366	2.107056961	1
5.502385039	1.404002608	1
7.432932365	4.236232628	1

Dataset for the implementation of SVM learning

DOI: https://doi.org/10.5281/zenodo.17608177

The visualization below provides a scatter plot of the dataset.
Training SVM with Gradient Descent

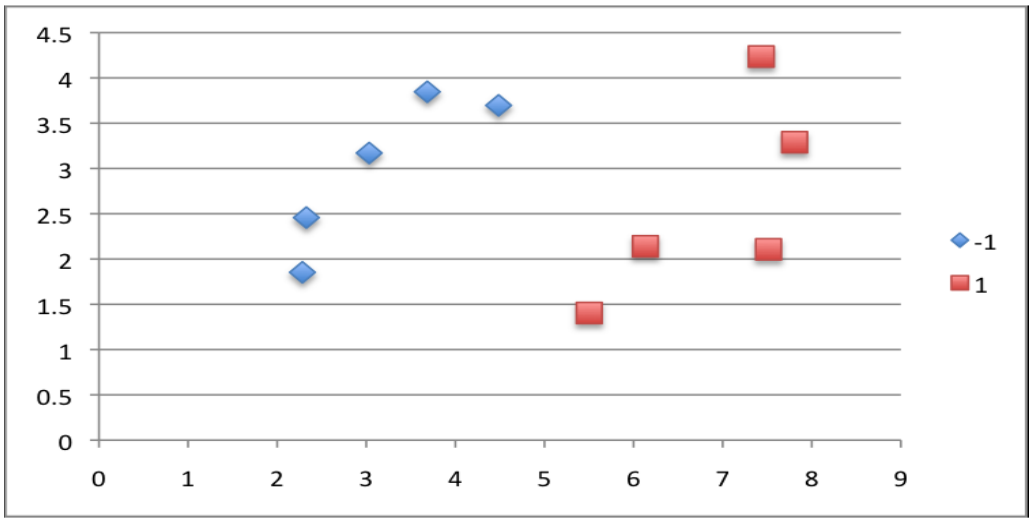


Figure1: SVM Dataset Scatter plot.

4.1 Training SVM with Gradient Descent

This section describes the form of the SVM model and how it can be learned using a variant of Stochastic gradient descent optimization procedure.

4.2 Form of Linear SVM Model

The Linear SVM model is a line and the goal of the learning algorithm is to find values for the coefficients that best separates the classes. The line is typically in the form (grouping the terms for readability)

$$B0 + (B1 \times X1) + (B2 \times X2) = 0 \quad (7)$$

where $B_0, B_1 \, and \, B_2$ are the coefficients and $X_1 \, and \, X_2$ are the input variables. This will be the form of the equation that we will be using with one small modification, we will drop the bias term (B0) also called the offset or the intercept. For example B0 = 0.

$$(B1 \times X1) + (B2 \times X2) = 0 \quad (8)$$

This means that the line will pass through the origin $(X_1 = 0 \, and \, X_2 = 0)$. This is just to make the implementation steps easier to follow and because our simple problem does not really need it, you can add the bias term back in if you like.

4.3 SVM Optimization Method

The optimization algorithm to find the coefficients can be stated as a quadratic programming problem. This is a type of constraint optimization where fast solvers can be used. We will not

be using this approach in this tutorial. Another approach that can be used to discover the coefficient values for Linear SVM is sub-gradient descent. In this method a random training pattern is selected each iteration and used to update the coefficients. After a large number of iterations (thousands or hundreds of thousands) the algorithm will settle on a stable set of coefficients. The coefficient update equation works as follows. First an output value is calculated as:

$$output = Y \times (B1 \times X1) + (B2 \times X2) \quad (9)$$

Two different update procedures are used depending on the output value. If the output value is greater than 1 it suggests that the training pattern was not a support vector. This means that the instance was not directly involved in calculating the output, in which case the weights are slightly decreased:

$$b = \left(1 - \frac{1}{t}\right) \times b \quad (10)$$

Where b is the weight that is being updated (such as B_1 *or* B_2), t is the current iteration (e.g. 1 for the first update, 2 for the second and so on). If the output is less than 1 then it is assumed that the training instance is a support vector and must be updated to better explain the data.

$$b = \left(1 - \frac{1}{t}\right) \times b + \frac{1}{lambda \times t} \times (y \times x) \quad (11)$$

Where b is the weight that is being updated, t is the current iteration and lambda is a parameter to the learning algorithm. The lambda is a learning parameter and is often set to very small values such as 0.0001 or smaller. The procedure is repeated until the error rate drops to a desirable level or for a very large fixed number of iterations. Smaller learning rates ofter require much longer training times. The number of iterations is a downside to this learning algorithm.

4.4 Learn an SVM Model from Training Data

In this section we will work through a few updates to the coefficients to demonstrate the SVM learning algorithm. We will use a very large lambda value: lambda = 0.45. This is unusually large and will force a lot of change on each update. Normally lambda values are very small.

4.5 **Learning Iteration #1**

We will start by setting the coefficients to 0.0.

$$B1 = 0.0$$
$$B2 = 0.0 \quad (12)$$

We also need to keep track of which iteration we are on: t = 1. We will train the model using the order of the training patterns. Ideally, the order of the patterns would be randomized to avoid the learning algorithm getting stuck. The first training pattern we will use to update the coefficients is:

DOI: https://doi.org/10.5281/zenodo.17608177

Instance: X1 = 2.327868056, X2 = 2.458016525, Y = -1. We can now calculate the output value for this iteration.

$$\begin{aligned} output &= Y \times (B1 \times X1) + (B2 \times X2) \\ &= -1 \times (0.0 \times 2.327868056) + (0,0 \times 2.458016525) \\ &= 0.0 \end{aligned}$$

Easy enough, the output is less than 1.0, therefore we will use the more complex update procedure (equation 11) that assumes the training pattern is a support vector:

$$b = \left(1 - \tfrac{1}{t}\right) \times b + \tfrac{1}{lambda \times t} \times (y \times x)$$

$$\begin{aligned} B1 &= \left(1 - \tfrac{1}{1}\right) \times 0.0 + \frac{1}{0.5 \times 1} \times (-1 \times 2.327868056) \\ &= -5.173040124 \end{aligned}$$

And for B2 this is:

$$\begin{aligned} B2 &= \left(1 - \tfrac{1}{1}\right) \times 0.0 + \frac{1}{0.5 \times 1} \times (-1 \times 2.458016525) \\ &= -5.462258944 \end{aligned}$$

4.6 Learning Iteration #2

We now have updated coefficients that we can use on the next iteration of the learning Algorithm with the second instance from the training dataset: Instance: X1 = 3.032830419, X2 = 3.170770366, Y = -1. Again, we must keep track of the iteration: t = 2. Let's repeat the process.

B1 = -5.173040124
B2 = -5.462258944

The output value is calculated as:

$$\text{Output} = -1 \times (-5.173040124 \times 3.032830419) + (-5.462258944 \times 3.170770366)$$

$$= 33.00852224$$

The output value is larger than 1.0, suggesting that this training instance is not a support vector. We can update the B1 coefficient accordingly:

$$b = \left(1 - \tfrac{1}{t}\right) \times b$$

$$B1 = \left(1 - \tfrac{1}{2}\right) \times -5.173040124$$

$$B1 = -2.586520062$$

And for the B2 coefficient:

$$B2 = \left(1 - \tfrac{1}{2}\right) \times -5.462258944$$
$$B2 = -2.731129472$$

4.7 More Iterations

We repeat this process for the rest of the dataset. One pass through the dataset is called an epoch. Now repeating the process for a further 15 epochs for a total of 160 iterations (16 epochs $\times 10$ updates per epoch). It is possible to keep track of the loss or the accuracy of the model for each epoch. This is a great way to get insight into whether the algorithm is converging or whether there is a bug in the implementation. Plotting the accuracy for the model at the end of each epoch, we achieved the following plot:

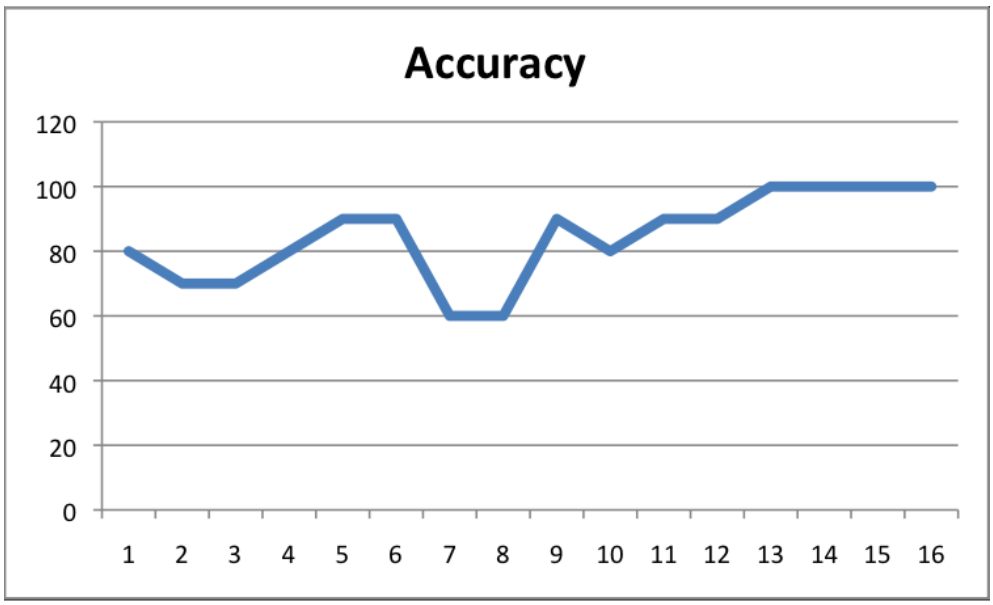


Figure 2: Support Vector Machine Model Accuracy.

From the plot, after16 epochs then we achieved an accuracy rate of 100% on the training data. We therefore arrive at final values for the coefficients as following:

B1 = 0.552391765
B2 = -0.724533592

Now that we have coefficients for the line, we can make predictions. In this section we will make predictions for the training data, but this could just as easily be adapted to make predictions for new data. Predictions can be made using the following equation:

$$output = (B1 \times X1) + (B2 \times X2)$$

The form of the learned hyper-plane is therefore:

$$0 + (0.552391765 \times X1) + (-0.724533592 \times X2) = 0$$

DOI: https://doi.org/10.5281/zenodo.17608177

4.8 Make Predictions with SVM Model

$$Y = -1 \; if \; output < 0$$
$$Y = +1 \; if \; output > 0$$

Using the above coefficients and these prediction rules, we can make a prediction for each instance in the training dataset as follows:

Output	Crisp	Y
-0.495020399	-1	-1
-0.622019096	-1	-1
-0.20066956	-1	-1
-0.75170826	-1	-1
-0.081298299	-1	-1
1.928999147	1	1
1.836907801	1	1
2.624496306	1	1
2.022225129	1	1
1.036597783	1	1

Listing 2:Accuracy of the Learned SVM Model.

Comparing the crisp prediction (Crisp) to the expected output column (Y), we can see that our model has achieved 100% accuracy.

5.0 Summary, Discussion and Conclusion

5.1 Summary:

In this paper we have empirically shown how to implement the Support Vector Machine learning algorithm using the stochastic sub-gradient descent optimization technique.
We learned the following:
a. How to update coefficient values for SVM using sub-gradient descent.
b. How to iterate the sub-gradient descent procedure to find good coefficient values.
c. How to make predictions using a learned SVM model.
We now know how to implement the Support Vector Machine algorithm from scratch using Sub gradient descent. This concludes our SVM implementation to nonlinear machine learning algorithms.

5.2 Discussion

The SSGD-based implementation of SVM significantly reduces computational complexity from $o(n^3)$ in quadratic solvers to $o(n)$ in stochastic updates. The sub-gradient approach handles non-differentiability of the hinge loss effectively. However, the convergence rate depends heavily on hyper-parameter tuning, especially learning rate scheduling and regularization strength.

5.3 Conclusion

This paper presents a detailed, step-by-step methodology for implementing SVM using Stochastic Sub-Gradient Descent. The approach combines the theoretical rigor of SVMs with the computational efficiency of stochastic optimization. Future research may explore adaptive learning rate schedules, kernel extensions, and hybridization with deep neural architectures for non-linear tasks.

References

1. Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT*, 177–186.
2. Chang, C.-C., & Lin, C.-J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 1–27.
3. Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.
4. Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., & Sundararajan, S. (2008). A dual coordinate descent method for large-scale linear SVM. *ICML*.
5. Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. *Microsoft Research Technical Report*.
6. Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2011). Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1), 3–30.
7. Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

DOI: *https://doi.org/10.5281/zenodo.17608177*