SVM On Micro Controller

Sameer Sharma (B21CS066), Shalin Jain (B21CS070)
Github

May 2, 2024

1 Introduction

The study of machine learning algorithms on embedded systems has gain significant attention in modern world in autonomous devices across the world. In this project we are going to implement **Support Vector Machine** [1] algorithm on **STM32 Microcontroller** board and going to explore its functionalities over multiple kernels.

2 Work Flow

2.1 Algorithm

The mathematical intuition behind the SVMs is that we try to find a hyperplane in multidimensional space between data-points belonging to various classes.

Hyper Plane:- In a binary classification problem a hyper-plane is a subspace which divides the space into two halves each containing one class.

Maximising Margins:- SVM aims to find a hyperplane which maximises the distance between class data-points. The closest data points are known as *support vectors*.

Equation of Hyperplane

$$\mathbf{w^T x} + b = 0$$

- $\mathbf{w}$ is normal vector to hyper plane.
- $\mathbf{x}$ is data point.
- **b** is the bias term.

Distance to Hyper Plane:- The distance of a data point $\mathbf{x}$ to the hyperplane can be calculated as the projection of $\mathbf{x}$ onto the normal vector $\mathbf{w}$, divided by the magnitude of $\mathbf{w}$.

Equation for distance

$$\mathbf{d} = \frac{|\mathbf{w^T x + b}|}{|\mathbf{w}|}$$

Margin Calculation:- The margin is the distance between two hyperplanes. The goal is to find optimal value of **w** and **b** such that the distance between the clusters of classes is maximized.

Kernel Trick:- If the data is not linearly separable then the data is projected to higher dimensional space. The inner product in this higher dimensional space corresponds to evaluation of kernel function. This led us to exploration of **Soft Margin SVM**.

2.2 Soft Margin SVM

This section discusses the math behind Soft Margin SVM [2]. Let the equation of positive hyperplane be

$$\pi^+ : \mathbf{w^T} \cdot \mathbf{x} + b = 0$$

Let the equation of the negative hyperplane be

$$\pi^- : \mathbf{w^T} \cdot \mathbf{x} + b = 0$$

Hence the margin between the hyperplane is

$$\mathbf{margin} = \frac{2}{\|w\|}$$

For SVMs we can write the constraint equations for optimal $\mathbf{w}^*, \mathbf{b}^*$ as follows:-

$$\mathbf{w}^*, \mathbf{b}^* = \text{argmin}_{w,b} \frac{\|w\|}{2} + C \cdot \frac{1}{n} \sum_{i=1}^{n} \mathcal{E}_i \tag{1}$$

- $\mathcal{E}$ = some units of distance from the correct hyperplane in the incorrect direction.
- For correctly classified points $\mathcal{E}_i = 0$ i.e. if $\mathbf{y_i}(\mathbf{w^T} \cdot \mathbf{x} + \mathbf{b}) \geq \mathbf{1}$
- For incorrectly classified points $\mathcal{E}_i > 0$
- Equal to 1 for **Support Vectors**

Equation 1 can be equivalently written as follows

$$\alpha^* = \max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2}$$

Here $\mathbf{x_i^T x_j}$ can be replaced by any kernel function for better results over non-linear separable datasets.

2.2.1 Kernels

1. Polynomial Kernel (Poly):-

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = (a + x_i^T \cdot x_j)^b$$

2. Radial Bias Kernel Function (RBF):-

$$\mathbf{K}(\mathbf{x_i}, \mathbf{x_j}) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

3 Datasets

Given is the list of datasets used for testing our algorithm

Dataset	Number of Features	Number of Unique Classes	Number of Samples
Iris Dataset	4	3	150
Bank Note Dataset	4	2	178
Circles	2	2	100
Moons	2	2	100

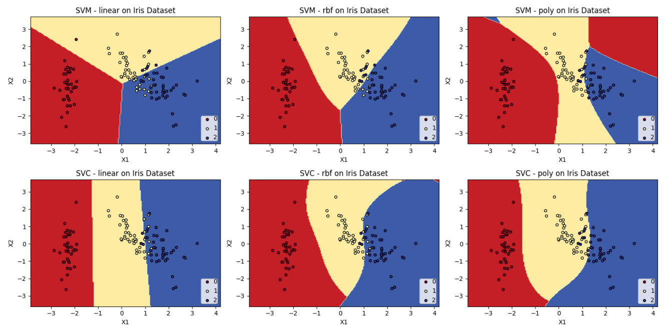
4 Testing and Results

Below is a table which shows the results of the results of the SVM and inbuilt SVC(SVM) on all datasets

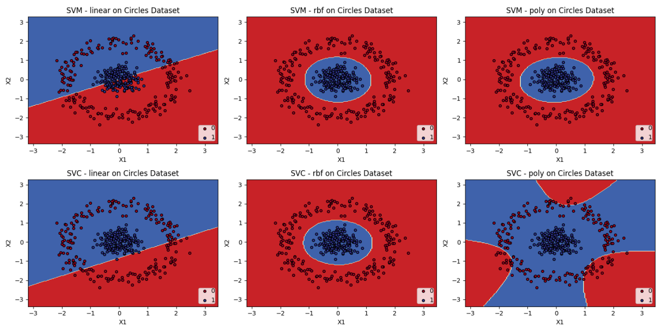
Results				
Dataset	**Kernel**	**SVM (%)**	**SVC(%)**	**Time (s)**
Iris	Linear	73.33	96.67	0.000996
	Rbf	90.00	96.67	0.000999
	Poly	100.00	100.00	0.000000
Circles	Linear	35.00	60.00	0.000997
	Rbf	100.00	100.00	0.000216
	Poly	100.00	65.00	0.000000
Moons	Linear	95.00	85.00	0.000000
	Rbf	85.00	100.00	0.000000
	Poly	85.00	85.00	0.000998
Bank Note	Linear	97.22	97.22	0.000000
	Rbf	88.89	100.00	0.011603
	Poly	100.00	97.22	0.000998

Table 1: Accuracy and Time Comparison

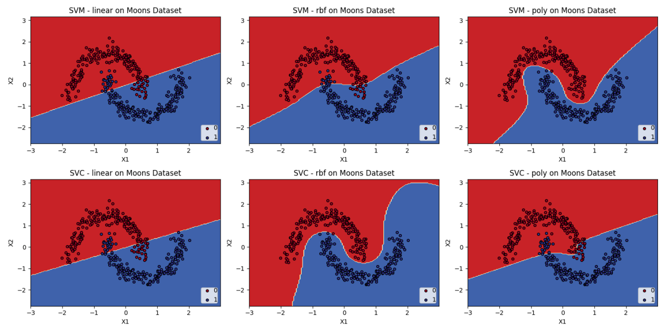
The below results clearly show that our SVM has performed better in many cases than the traditional or inbuilt SVM(SVC) and also given faster results.



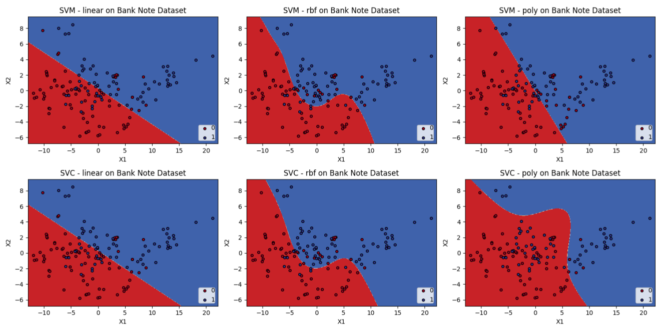
(a) Iris



(b) Circles

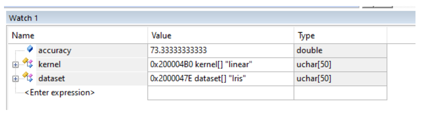


(c) Moons



(d) Bank Note

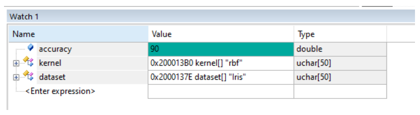
Figure 1: Different datasets with different kernels



Watch 1

Name	Value	Type
accuracy	73.33333333333	double
kernel	0x200004B0 kernel[] "linear"	uchar[50]
dataset	0x2000047E dataset[] "Iris"	uchar[50]
<Enter expression>		

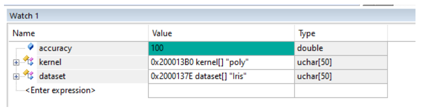
(a) iris-linear



Watch 1

Name	Value	Type
accuracy	90	double
kernel	0x200013B0 kernel[] "rbf"	uchar[50]
dataset	0x2000137E dataset[] "Iris"	uchar[50]
<Enter expression>		

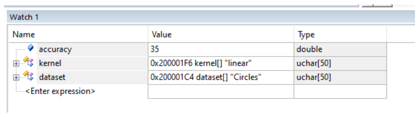
(b) iris - rbf



Watch 1

Name	Value	Type
accuracy	100	double
kernel	0x200013B0 kernel[] "poly"	uchar[50]
dataset	0x2000137E dataset[] "Iris"	uchar[50]
<Enter expression>		

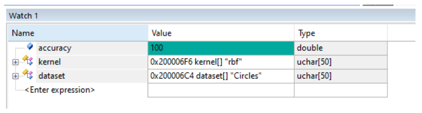
(c) iris - poly



Watch 1

Name	Value	Type
accuracy	35	double
kernel	0x200001F6 kernel[] "linear"	uchar[50]
dataset	0x200001C4 dataset[] "Circles"	uchar[50]
<Enter expression>		

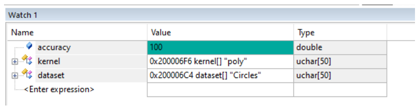
(d) circles-linear



Watch 1

Name	Value	Type
accuracy	100	double
kernel	0x200006F6 kernel[] "rbf"	uchar[50]
dataset	0x200006C4 dataset[] "Circles"	uchar[50]
<Enter expression>		

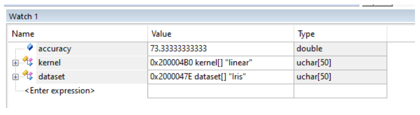
(e) iris - rbf



Watch 1

Name	Value	Type
accuracy	100	double
kernel	0x200006F6 kernel[] "poly"	uchar[50]
dataset	0x200006C4 dataset[] "Circles"	uchar[50]
<Enter expression>		

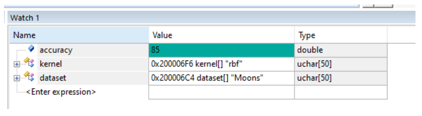
(f) circles - poly



Watch 1

Name	Value	Type
accuracy	73.33333333333	double
kernel	0x200004B0 kernel[] "linear"	uchar[50]
dataset	0x2000047E dataset[] "Iris"	uchar[50]
<Enter expression>		

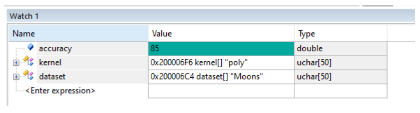
(g) iris-linear



Watch 1

Name	Value	Type
accuracy	85	double
kernel	0x200006F6 kernel[] "rbf"	uchar[50]
dataset	0x200006C4 dataset[] "Moons"	uchar[50]
<Enter expression>		

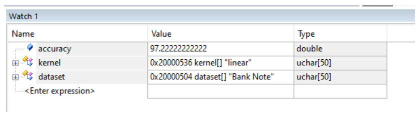
(h) moons - rbf



Watch 1

Name	Value	Type
accuracy	85	double
kernel	0x200006F6 kernel[] "poly"	uchar[50]
dataset	0x200006C4 dataset[] "Moons"	uchar[50]
<Enter expression>		

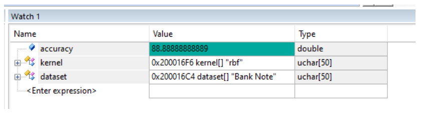
(i) moons - poly



Watch 1

Name	Value	Type
accuracy	97.22222222222	double
kernel	0x20000536 kernel[] "linear"	uchar[50]
dataset	0x20000504 dataset[] "Bank Note"	uchar[50]
<Enter expression>		

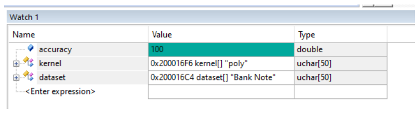
(j) banknote - linear



Watch 1

Name	Value	Type
accuracy	88.88888888889	double
kernel	0x200016F6 kernel[] "rbf"	uchar[50]
dataset	0x200016C4 dataset[] "Bank Note"	uchar[50]
<Enter expression>		

(k) banknote - rbf



Watch 1

Name	Value	Type
accuracy	100	double
kernel	0x200016F6 kernel[] "poly"	uchar[50]
dataset	0x200016C4 dataset[] "Bank Note"	uchar[50]
<Enter expression>		

(l) banknote - poly

Figure 2: Accuracies

5 Usage

5.1 Training

1. Initialize an instance of the SVM class. You may need to define the kernel for your model (Linear kernel is default)

2. Select and preprocess the dataset (Standard Scaling is preferred). The format of the dataset should be two numpy arrays, X and y containing features and labels respectively.

3. Now, train the model using $model.fit(X, y)$
4. For prediction, use $model.predict(X\_sample)$

5.2 Conversion (Python to C)

1. Initialize an instance of the Converter class.
2. Now, you can convert you python model to a C header file by running the *Converter.convert* function where the inputs are the python model, training and test data and the standard scaler used for preprocessing the training data and finally the precision.

5.3 Run on STM32

1. Include the C header file generated in the above subsection in the model.c file.
2. Now build the project and execute it on stm32 as usual.

6 Innovations

6.1 Custom SVM class

Implemented a custom SVM class in python to train the model on the general purpose machine and get the optimal weights. Also, to predict the class labels on STM32, the same class is also implemented in C containing different kernels as Linear, Poly and RBF kernels whose description is given above. This custom class can be used to train a SVM model on any dataset provided it fits inside the small memory of the microcontroller. The size of the model depends on the size of dataset which can be found from the given eqautions in *bytes*.

Equation for Linear Kernel

$$(2 * n_{features} + n_{classes} * n_{features} + 2 * n_{classes} + n_{features} * n_{samples} + n_{samples}) * 16 + c \tag{3}$$

Equation for RFB and Poly Kernel

$$(2 * n_{features} + n_{classes} * n_{features} + 2 * n_{classes} + n_{features} * n_{samples} + n_{samples} + n_{features} * n_{train-samples}) * 16 + c \tag{4}$$

Where,

- $\mathbf{n_{features}}$ is the number of features in the dataset.
- $\mathbf{n_{classes}}$ is number of unique classes in the dataset.
- $\mathbf{n_{samples}}$ is number of testing samples.
- $\mathbf{n_{train-samples}}$ is number of training samples.
- **c** is the constant sapce utilized by the C code. It is around 7kB.

6.2 Converter

We implemented a class in python to deploy the model trained from the python class on the microcontroller. It takes an instance of the Custom SVM class in input and generates a c header file which contains the weights, biases, mean, standard deviation and other hyper parameters related to the kernels. We can also store the test samples in the header file to assess the accuracy of the model on STM32.

We also introduced a term precision to fit large models on the microcontroller by truncating weights to the specified decimal places. Although, it is a tradeoff since decreasing the precision also decreases the accuracy but makes the model fit on the hardware.

References

[1] L. Galletti, "Support vector machines," *Medium*, unknown. [Online]. Available: https://medium.com/@gallettilance/support-vector-machines-16241417ee6d

[2] K. Datta, "A complete guide to support vector machines (svms)," *Medium*, unknown. [Online]. Available: https://medium.com/@kushaldps1996/a-complete-guide-to-support-vector-machines-svms-501e71aec19e