AI2612 Machine Learning Project Report
Implementation of SVM and Exploration of Kernel Functions

Huayi Wang
522030910116

Abstract

Support Vector Machines (SVM) are powerful supervised learning models widely used for classification and regression tasks. Despite their maturity and widespread implementation, developing SVM from scratch allows for a deeper understanding of their inner workings. In this project, we implement SVM from scratch and explore various kernel functions, including Sequential Minimal Optimization (SMO) algorithm, the kernel tricks, the Support Vector Classifier (SVC) for multi-class classification, and Multiple Kernel Learning (MKL). We conduct experiments on the MNIST and CIFAR-10 datasets to evaluate the performance of different choices.

1. Introduction

Support Vector Machines (SVM) are powerful supervised learning models widely used for classification, regression, and other tasks.

The core idea of SVM is to find a hyperplane that best separates the data into different classes by maximizing the margin. To handle high-dimensional and non-linearly separable data, SVM employs the kernel trick, which maps the input data into a higher-dimensional feature space. To speed up computation, the Sequential Minimal Optimization (SMO) algorithm provides a more efficient way of solving the dual problem arising from the derivation of SVM. Support Vector Classification (SVC) extends SVM to handle multi-class classification problems using two common strategies: One-vs-One (OvO) and One-vs-Rest (OvR). Additionally, Multiple Kernel Learning (MKL) enhances SVM's performance and robustness by applying and optimizing the combination of multiple kernels.

In this project, we implement SVM from scratch without relying on any pre-existing machine learning libraries. Our implementation encompasses: linear SVM, extension to non-linear SVM with kernel trick, further extension to SVC, SMO algorithm to improve the efficiency and MKL techniques to boost performance.

Finally, we carry out experiments on the MNIST and CIFAR-10 datasets to evaluate the performance of different kernels and analyze the impact of various parameters.

2. Methods

In this section, we will do necessary derivation of relevant algorithms and describe our methods.

2.1. Data Processing

For the MNIST dataset, we use the original 28×28 pixel images as features.

For the CIFAR-10 dataset, in addition to using the original $3 \times 32 \times 32$ pixel images as features, we also implement ResNet18, a widely used convolutional neural network (CNN), for image classification. We then extract the output of the average pooling layer as features.

2.2. Linear SVM and Kernel Trick

Giving a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ is the feature and $\boldsymbol{x}_i \in \mathbb{R}^n$ is the label. The primal optimization problem for non-linearly separable datasets can be formulated as

$$\begin{aligned} \min_{\boldsymbol{w},b} \ & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^m \xi_i \\ \text{s.t.} \ & y_i\left(\boldsymbol{w}^T\boldsymbol{x}_i + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0, \quad i = 1, 2, \ldots, m \end{aligned} \tag{1}$$

Here $\boldsymbol{w} \in \mathbb{R}^n$ is the weight vector, $b \in \mathbb{R}$ is the bias term, $\xi_i \geq 0$ is the slack variable and C is the penalty parameter. The dual problem can be formulated as

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \ & \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m \alpha_i\alpha_j y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j \\ \text{s.t.} \ & \sum_{i=1}^m \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, m \end{aligned} \tag{2}$$

This a quadratic programming problem. In our implementation, we adopt Sequential Minimal Optimization (SMO) algorithm to solve this dual problem efficiently, which will be introduced in Section 2.3.

To extend linear SVM to the non-linear one, let ϕ denotes the mapping function, then $\boldsymbol{x}_i^T \boldsymbol{x}_j$ in the optimization goal can be replaced with $\phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$, which can futher be replaced with kernel function $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$. Then the dual problem becomes

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ \text{s.t.} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & 0 \le \alpha_i \le C, \quad i = 1, 2, \ldots, m \end{aligned} \tag{3}$$

Kernel functions enable solving nonlinear problems by implicitly mapping data into higher-dimensional spaces, while allowing computation to remain efficient in the original input space. We implement four common kernels as shown in Table 1.

Kernel	$K(\boldsymbol{x}, \boldsymbol{y})$
Linear	$\boldsymbol{x}^T \boldsymbol{y}$
Gaussian	$\exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$
Polynomial	$(\gamma \boldsymbol{x}^T \boldsymbol{y} + r)^d$
Sigmoid	$\tanh(\gamma \boldsymbol{x}^T \boldsymbol{y} + r)$

Table 1. Kernel functions

2.3. Sequential Minimal Optimization (SMO)

The core idea of SMO is iteratively optimizing two variables at each step while fixing other variables. Suppose α_1 and α_2 are selected, then the optimization problem can be formulated as

$$\begin{aligned} \max_{\alpha_1, \alpha_2} \quad & \alpha_1 + \alpha_2 - \alpha_1 \alpha_2 y_1 y_2 K(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ & - \frac{1}{2} \alpha_1^2 y_1^2 K(\boldsymbol{x}_1, \boldsymbol{x}_1) - \frac{1}{2} \alpha_2^2 y_2^2 K(\boldsymbol{x}_2, \boldsymbol{x}_2) \\ & - \alpha_1 y_1 \sum_{i=3}^{m} \alpha_i y_i K(\boldsymbol{x}_1, \boldsymbol{x}_i) - \alpha_2 y_2 \sum_{i=3}^{m} \alpha_i y_i K(\boldsymbol{x}_2, \boldsymbol{x}_i) \\ \text{s.t.} \quad & \alpha_1 y_1 + \alpha_2 y_2 = -\sum_{i=3}^{m} \alpha_i y_i \\ & 0 \le \alpha_1, \alpha_2 \le C \end{aligned} \tag{4}$$

Decision boundary is $f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$. Let $e_i = f(\boldsymbol{x}_i) - y_i$ denotes the prediction error and $\eta = K(\boldsymbol{x}_1, \boldsymbol{x}_1) + K(\boldsymbol{x}_2, \boldsymbol{x}_2) - 2K(\boldsymbol{x}_1, \boldsymbol{x}_2)$. Then we can get the close solution of α_2:

$$\alpha_2' = \alpha_2 + \frac{y_2 (e_1 - e_2)}{\eta} \tag{5}$$

Since α is constrained in $[0, C]$, we need to clip α_2' to $[L, H]$, where

$$\begin{aligned} L &= \begin{cases} \max\{0, \alpha_2 + \alpha_1 - C\}, & y_1 = y_2 \\ \max\{0, \alpha_2 - \alpha_1\}, & y_1 \ne y_2 \end{cases} \\ H &= \begin{cases} \min\{C, \alpha_2 + \alpha_1\}, & y_1 = y_2 \\ \min\{C, \alpha_2 - \alpha_1 + C\}, & y_1 \ne y_2 \end{cases} \end{aligned} \tag{6}$$

Then α_1' can be updated by the constraint in Eq.4:

$$\alpha_1' = \alpha_1 + y_1 y_2 (\alpha_2 - \alpha_2') \tag{7}$$

Finally, we need to update the bias b of decision function as follow

$$b' = \begin{cases} b_1, & 0 < \alpha_1' < C \\ b_2, & 0 < \alpha_2' < C \\ \frac{1}{2}(b_1 + b_2), & \text{otherwise} \end{cases} \tag{8}$$

where b_1 and b_2 are calculated as

$$\begin{aligned} b_1 &= b - e_1 - y_1 K(\boldsymbol{x}_1, \boldsymbol{x}_1)(\alpha_1' - \alpha_1) \\ &\quad - y_2 K(\boldsymbol{x}_1, \boldsymbol{x}_2)(\alpha_2' - \alpha_2) \\ b_2 &= b - e_2 - y_1 K(\boldsymbol{x}_1, \boldsymbol{x}_2)(\alpha_1' - \alpha_1) \\ &\quad - y_2 K(\boldsymbol{x}_2, \boldsymbol{x}_2)(\alpha_2' - \alpha_2) \end{aligned} \tag{9}$$

The last remaining problem is how to select α_i and α_j at each step. A natural idea is random selection, but it tends to delay convergence due to many updates being insignificantly small. Conversely, always choosing ones that maximize the step size is computationally expensive. Hence, we adopt a balanced approach: in the outer loop, we traverse all samples, i.e., all α_i, and then select α_j maximizes the step size in the inner loop. As discussed in Section 3.3, this method proves to be quite efficient.

The overall algorithm is shown in Algorithm 1.

2.4. Support Vector Classification (SVC)

To extend binary SVM into multi-class classification, One-vs-One (OvO) and One-vs-Rest (OvR) are two typical strategies. OvO trains C_k^2 SVMs, where each one discriminate one class from another class. OvR trains k classifiers, where each one discriminate one class from the rest $k-1$ classes. Although OvO trains more SVMs, each SVM will be trained on a much smaller dataset with higher accuracy.

Algorithm 1 Sequential Minimal Optimization

Require: Training dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$
Initialization: $\boldsymbol{\alpha} \leftarrow \mathbf{0}, b \leftarrow 0$
Compute kernel matrix $K \in \mathbb{R}^{n \times n}$
while not converge **do**
  $\boldsymbol{\alpha}^{\text{old}} \leftarrow \boldsymbol{\alpha}$
  for $i = 1, 2, ..., |\mathcal{D}|$ **do**
    if KKT conditions are satisfied for α_i **then**
      continue
    end if
    Compute prediction error $e_i = f(\boldsymbol{x}_i) - y_i$
    Select α_j where $j = \arg\max_j |e_j - e_i|$
    Compute bounds L and H defined in Eq.6
    Update $\alpha_j \leftarrow \alpha_j'$ defined in Eq.5, clip to $[L, H]$
    Update $\alpha_i \leftarrow \alpha_i'$ defined in Eq.7
    Update bias $b \leftarrow b'$ defined in Eq.8
  end for
  if $|\boldsymbol{\alpha}^{\text{old}} - \boldsymbol{\alpha}| \leq \epsilon$ **then**
    break
  end if
end while

So we adopt this strategy in our implementation. As discussed in Section 3.3, this method proves to be more efficient than OvR. The training algorithm of OvO strategy is shown in Algorithm 2.

For prediction during testing, each classifier $C_{\{i,j\}}$ provides a prediction $y_{\{i,j\}}$. The final prediction is the class with the most votes, i.e.

$$\text{prediction} = \arg \max_{1 \leq i \leq k} \sum_{j \neq i} \mathbf{1}\left\{\boldsymbol{y}_{\{i,j\}} = i\right\} \tag{10}$$

Algorithm 2 One-vs-One strategy

Require: Training dataset $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^k$
Initialization C_k^2 SVMs $C_{\{i,j\}}$
for i, j in C_k^2 combinations **do**
  Select sub-dataset $\mathcal{D}^{\text{sub}} = \mathcal{D}_i \cup \mathcal{D}_j$
  Train SVM $C_{\{i,j\}}$ on $\mathcal{D}^{\text{sub}}$ defined in Algorithm 1
end for

2.5. Multiple Kernel Learning (MKL)

Multiple Kernel Learning (MKL) extends the idea of kernel methods by combining multiple kernels to improve the performance of SVM. We consider a set of M kernel matrix $\{K_k\}_{k=1}^M$, then the combined kernel K is a weighted sum of these individual kernels

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{M} \lambda_k K_k(\boldsymbol{x}, \boldsymbol{y}) \tag{11}$$

The corresponding dual problem is specified as

$$\begin{aligned} \max_{\boldsymbol{\alpha}, \boldsymbol{\lambda}} & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \sum_{k=1}^M \lambda_k K_k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) \\ \text{s.t.} & \sum_{i=1}^m \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, m \end{aligned} \tag{12}$$

The closed-form solution for $\boldsymbol{\lambda}$ is given by

$$\lambda_k' = \frac{g_k}{\sum_{i=1}^l g_i} \tag{13}$$

where $g_k^2 = \lambda_k^2 (\boldsymbol{\alpha}^T \boldsymbol{y})^T \boldsymbol{K}_k (\boldsymbol{\alpha}^T \boldsymbol{y})$.

To update $\boldsymbol{\lambda}$ and $\boldsymbol{\alpha}$, we first fix $\boldsymbol{\lambda}$ and update $\boldsymbol{\alpha}$ using a method similar to the SMO algorithm. Then we fix $\boldsymbol{\alpha}$ and update $\boldsymbol{\lambda}$. However, update $\boldsymbol{\lambda}$ every time $\boldsymbol{\alpha}$ changes can be time-consuming. Therefore, we adopt an approximation: we only update $\boldsymbol{\lambda}$ in the outer loop of Algorithm 1, i.e. only when $\boldsymbol{\alpha}$ has traversed all samples and experienced a significant change. Although this approximation isn't an online update, as discussed in Section 3.3, it significantly improves computational efficiency without substantially harming accuracy.

The overall multiple kernel learning algorithm is presented in Algorithm 3.

Algorithm 3 Multiple Kernel Learning

Require: Training dataset $\mathcal{D}$, kernel set $\{K_k\}_{k=1}^M$
Initialization: $\boldsymbol{\alpha} \leftarrow \mathbf{0}, b \leftarrow 0$
Initialization: kernel weights $\boldsymbol{\lambda} \leftarrow \frac{1}{M}(1, 1, ..., 1)^T$
while not converge **do**
  $\boldsymbol{\alpha}^{\text{old}} \leftarrow \boldsymbol{\alpha}$
  Compute kernel matrix K defined in Eq. 11
  for $i = 1, 2, ..., |\mathcal{D}|$ **do**
    Same to Line 6 - Line 14 in Algorithm 1
  end for
  Update $\boldsymbol{\lambda}$ defined in Eq.13
  if $|\boldsymbol{\alpha}^{\text{old}} - \boldsymbol{\alpha}| \leq \epsilon$ **then**
    break
  end if
end while

3. Experiments

In this section, we conduct various experiments to evaluate the effectiveness and the efficiency of our implementation across various datasets and kernel types.

3.1. Implementation Details

We conducted experiments on three datasets—MNIST, CIFAR-10, and CIFAR-10-features (introduced in Section 2.1)—to evaluate the accuracy of our implementation. For training, we selected 500 samples per class, resulting in a total of 5000 samples. For testing, we selected 100 samples per class, resulting in a total of 1000 samples.

The hyperparameter γ was selected through grid search, a method used to search through a specified subset of the hyperparameter space. In our experiments, we defined a range of values for γ and evaluated the model's performance for each value. The optimal γ values, which produced the highest accuracy, were chosen for each dataset and kernel type. These values are listed in Table 2.

Kernel	MNIST	CIFAR-10	CIFAR-10-features
Gaussian	0.01	0.008	0.05
Polynomial	0.03	0.02	0.04
Sigmoid	0.007	0.001	0.01

Table 2. Hyperparameter γ settings

Other parameters were fixed throughout the experiments: penalty $C = 3$, $r = 1$ and $d = 3$ for the polynomial kernel, and $r = -1$ for the sigmoid kernel.

3.2. Results

We first conducted experiments on the MNIST dataset, and the results are presented in Table 3. Overall, the four kernels tested achieved high accuracy, with at least 89% on the test set, and proved to be efficient, with training times capped at 140 seconds. This indicates that our SMO algorithm and multi-class classification implementation are both effective and efficient. Notably, the gaussian kernel achieved the highest test accuracy of 94.00%, while the polynomial kernel also performed admirably. In comparison, the linear kernel, despite achieving perfect training accuracy, performed the worst on the test set due to overfitting.

Kernel	**Train Acc**	**Test Acc**	**Train Time**
Linear	100.00%	89.80%	02:04
Gaussian	99.24%	**94.00%**	02:12
Polynomial	100.00%	93.90%	01:15
Sigmoid	96.74%	91.60%	01:26

Table 3. Performance of different kernels on MNIST

To further investigate the performance of our implementation, particularly in the context of multiple kernel learning (MKL), we conducted experiments using all possible kernel combinations on the CIFAR-10 dataset. The complete results are presented in Table 5.

From the first four single-kernel results, we observe that the performance on CIFAR-10 is generally worse than on MNIST, with an average accuracy of around 30%. This decrease in performance is likely due to the increased complexity of CIFAR-10 images, which have three color channels, making it more challenging for SVMs to capture relevant features. Despite this, the gaussian kernel still achieved a relatively high test accuracy of 43.2%. Additionally, the overall training time remained short, further validating the efficiency demonstrated in our MNIST results.

Examining experiments 5-14, where all kernel combinations were tested, reveals that SVMs benefit from multiple kernels, showing noticeable improvements in accuracy. For instance, in experiment 6, where linear and polynomial kernels were combined, the accuracy was improved to 41.00%, despite each kernel performing poorly on its own with accuracy of 31.60% and 31.30% respectively. Moreover, all training times were under 200 seconds, demonstrating that our MKL implementation is both effective and efficient.

To improve performance on CIFAR-10, we introduced CIFAR-10-features (introduced in Section 2.1). The comparison of results is presented in Table 4. By leveraging more informative representations of the CIFAR-10 images, our approach significantly boosts performance across all kernels.

Kernel	CIFAR-10	CIFAR-10-features
Linear	31.60%	78.20%
Gaussian	43.20%	**87.60%**
Polynomial	31.30%	86.30%
Sigmoid	21.90%	81.30%

Table 4. Test accuracy of different kernels on CIFAR-10 and CIFAR-10-features

3.3. Ablation Studies

Choices of α_i and α_j In Section 2.3, we proposed three methods for selecting α_i and α_j:

- *Random*: Both α_i and α_j are selected randomly. This method has a time complexity of $O(1)$ but requires many iterations to converge.
- *Max Stepsize*: Always selects α_i and α_j to maximize $|e_i - e_j|$. This method has a time complexity of $O(n^2)$ but converges in fewer iterations.
- *Balanced (ours)*: Traverses α_i (equivalent to randomly select α_i) and then selects α_j to maximize $|e_i - e_j|$. This method has a time complexity of $O(n)$ and converges in a moderate number of iterations.

We experimented with these three options on CIFAR-10 using gaussian and polynomial kernels. The results, including accuracy and training time, are presented in Table

Index	Linear	Gaussian	Polynomial	Sigmoid	Train Acc	Test Acc	Train Time	Test Time
1	✓				91.72%	31.60%	04:37	00:00
2		✓			98.12%	**43.20%**	01:48	00:01
3			✓		55.10%	31.30%	02:15	00:01
4				✓	25.54%	21.90%	00:57	00:00
5	✓	✓			100.00%	**45.60%**	02:12	00:02
6	✓		✓		62.80%	41.00%	02:51	00:02
7		✓	✓		100.00%	45.10%	02:45	00:02
8		✓		✓	100.00%	44.90%	02:32	00:02
9			✓	✓	61.70%	39.60%	02:55	00:02
10	✓	✓	✓		100.00%	45.00%	02:55	00:03
11	✓	✓		✓	100.00%	45.00%	03:00	00:03
12	✓		✓	✓	62.46%	41.40%	03:16	00:03
13		✓	✓	✓	100.00%	44.90%	02:53	00:03
14	✓	✓	✓	✓	100.00%	44.80%	02:51	00:04

Table 5. Performance of different kernel combinations on CIFAR-10

6. Note that the training time for the *Max Stepsize* method was estimated using the `tqdm` library, and we did not complete the entire training due to the time-consuming training. From the results, we observe few differences in test accuracy among the methods. However, there is a significant difference in training time, with our *balanced* method being the most efficient.

Kernel	Method	Train Time	Test Acc
Gaussian	Random	02:42	42.80%
	Max Stepsize	1:12:30	*None*
	Balanced (ours)	**01:48**	43.20%
Polynomial	Random	03:12	31.40%
	Max Stepsize	1:37:24	*None*
	Balanced (ours)	**02:15**	31.30%

Table 6. Comparison between three choices of α_i and α_j

Choices of multi-classification strategy In Section 2.4, we propose two options on multiple classification strategy: One-vs-One (OvO) and One-vs-Rest (OvR). Similar to previous section, we also experimented with these two strategies on CIFAR-10 using gaussian and polynomial kernels. The results, shown in Table 7, indicate that OvO achieves higher test accuracy and requires less training time compared to OvR.

Kernel	Strategy	Train Time	Test Acc
Gaussian	OvO	**01:48**	**43.20%**
	OvR	13:18	39.50%
Polynomial	OvO	**02:15**	**31.30%**
	OvR	16:05	23.90%

Table 7. Comparison of different multi-classification strategies

Methods of updating multiple kernel weights In Section 2.5, we proposed two methods for updating multiple kernel weights $\boldsymbol{\lambda}$:

- *Online*: Update $\boldsymbol{\lambda}$ in the inner loop, i.e. update as soon as $\boldsymbol{\alpha}$ updates.
- *Periodic (ours)*: Update $\boldsymbol{\lambda}$ in the outer loop, i.e. update when $\boldsymbol{\alpha}$ has a significant change. This method may introduce some bias but updates much less frequently than the previous method.

Additionally, we introduce another baseline, *Heuristic*, which first trains separate single-kernel SVMs. Then, kernel weights are assigned proportionally based on the corresponding prediction accuracy on the training set.

The experiment setting are same to previous two ablation studies and the results are presented in Table 8. We observe that our *Periodic* method significantly reduces training time complexity compared to the *Online* method without compromising test accuracy. It also achieves higher accuracy compared to the *Heuristic* method. Therefore, our method is both efficient and effective.

Kernel	Method	Train Time	Test Acc
Gaussian	Online	1:53:21	**43.80%**
	Periodic (ours)	**01:48**	**43.20%**
	Heuristic	02:02	25.20%
Polynomial	Online	2:13:05	**31.50%**
	Periodic (ours)	**02:15**	**31.30%**
	Heuristic	02:30	23.50%

Table 8. Comparison of different methods for updating multiple kernel weights

4. Conclusion

In this report, we have presented a comprehensive overview of Support Vector Machines (SVMs), covering their derivation, implementation, and evaluation. The results underscore the effectiveness and efficiency of our implementation. Developing SVMs from scratch has enabled us to gain a deeper understanding of their inner workings and practical applications.

5. Acknowledgements

The formula derivations in Section 2.2, 2.3, 2.4, 2.5 are largely attributed to Xiangyuan Xue (21st senior) and Stanford CS229 lecture notes. Thanks!