AI2611 Machine Learning Project Report Support Vector Machine From Scratch

Xiangyuan Xue (521030910387)
School of Electronic Information and Electrical Engineering

I. Introduction

Support Vector Machine (SVM) [4] is a supervised machine learning algorithm for classification, regression and other tasks. Cortes and Vapnik first proposed SVM in 1995. In 1998, Platt proposed Sequential Minimal Optimization (SMO) [7] to accelerate the training process. In 2004, Lanckriet et al. proposed Multiple Kernel Learning (MKL) [8] to combine multiple kernels and improve the performance. Later, many methods were proposed to improve and accelerate MKL, such as MKL-SMO [11] and SimpleMKL [12]. Nowadays, SVM has become mature and widespread. Libraries such as LIBSVM [5] and scikit-learn [6] include the implementation of SVM and achieve relatively excellent performance.

In this project, we will implement SVM from scratch without using any machine learning libraries. We will implement the linear SVM and then extend it to non-linear with kernel trick. We will also implement SMO algorithm to improve the efficiency and apply MKL algorithm to enhance the robustness. Finally, we will test the performance of our implementation including efficiency and accuracy and research on the influence of different parameters. In section II, we will derive linear SVM, kernel trick, SMO algorithm and MKL algorithm. We will describe our algorithm implementation and multi-class strategy. In section III, we will carry out experiments on the MNIST dataset and the CIFAR10 dataset. We will report the results and present our analysis. In section IV, we will summarize our conclusions.

II. Algorithm

A. Support Vector Machine

Given a training set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, where $\boldsymbol{x}_i \in \mathbb{R}^n$ is the feature vector and $y_i \in \{+1, -1\}$ is the label. The linear SVM expects to find a hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ to separate the positive samples and the negative samples, where $\boldsymbol{w} \in \mathbb{R}^n$ is the normal vector and $b \in \mathbb{R}$ is the bias. For any sample $\boldsymbol{x}$, the distance from $\boldsymbol{x}$ to the hyperplane is specified as

$$r = \frac{|\boldsymbol{w}^T\boldsymbol{x} + b|}{\|\boldsymbol{w}\|}$$

Suppose all the samples are correctly classified. For any sample $\boldsymbol{x}_i$, it holds that

$$\begin{cases} \boldsymbol{w}^T\boldsymbol{x}_i + b \geq +1, & y_i = +1 \\ \boldsymbol{w}^T\boldsymbol{x}_i + b \leq -1, & y_i = -1 \end{cases}$$

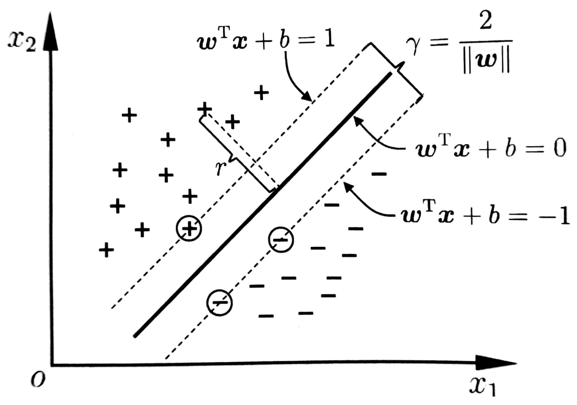
x_2
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=1$
$\gamma = \frac{2}{\|\boldsymbol{w}\|}$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=-1$
r
o
x_1

Fig. 1. Support Vector Machine

Notice that some samples satisfies $\boldsymbol{w}^T\boldsymbol{x}_i + b = \pm 1$, which are called support vectors. The distance from a support vector to the hyperplane is specified as

$$r = \frac{1}{\|\boldsymbol{w}\|}$$

which is so-called margin. The goal is to maximize the margin, namely to minimize $\|\boldsymbol{w}\|$. Hence, the optimization problem is formulated as

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

However, the samples are not always linearly separable, so hinge loss is introduced to penalize the mistakes where

$$\ell_{\text{hinge}}(z) = \max\{0, 1 - z\}$$

Then the optimization problem is formulated as

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^m \ell_{\text{hinge}}[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b)]$$

If we introduce slack variables $\xi_i \geq 0$, the optimization problem can be rewritten as

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

To find the dual problem, we introduce Lagrange multipliers $\alpha_i \geq 0$ and $\mu_i \geq 0$. The Lagrangian is specified as

$$\begin{aligned}\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\mu}) &= \frac{1}{2}\left\|\boldsymbol{w}\right\|^2 + C\sum_{i=1}^{m}\xi_i \\ &- \sum_{i=1}^{m}\alpha_i[y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{m}\mu_i\xi_i\end{aligned}$$

Take the partial derivatives with respect to $\boldsymbol{w}$, b and ξ_i as zero, it holds that

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}} &= \boldsymbol{w} - \sum_{i=1}^{m}\alpha_i y_i \boldsymbol{x}_i = 0 \\ \frac{\partial\mathcal{L}}{\partial b} &= -\sum_{i=1}^{m}\alpha_i y_i = 0 \\ \frac{\partial\mathcal{L}}{\partial\xi_i} &= C - \alpha_i - \mu_i = 0\end{aligned}$$

Hence, the dual problem can be formulated as

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

which is a quadratic programming problem. Methods such as gradient descent and coordinate descent can be applied to solve this optimization problem efficiently [1].

B. Kernel Trick

Sometimes the samples are not linearly separable, but they may be linearly separable in a higher dimensional space. Let ϕ denote the mapping function, then a sample $\boldsymbol{x}_i$ will be mapped to $\phi(\boldsymbol{x}_i)$. Then the dual problem can be formulated as

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

It is interesting that we do not need to know the mapping function ϕ explicitly. Instead, we define a kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$ which contains enough information for computation in the dual problem. Then the optimization problem can be rewritten as

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

Such technique avoids computation in a higher dimensional space, which is called kernel trick. Note that the kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ should satisfy Mercer's condition. Several common kernel functions are listed in following table. Note that γ, r and d are kernel parameters. Lienar kernel is simple and fast, while Gaussian kernel has strong representation ability. They are the most commonly used kernel functions. Nonetheless, we will implement and compare all the kernel functions listed in the table in our experiments.

TABLE I
KERNEL FUNCTION

Kernel Function	$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$
Linear Kernel	$\boldsymbol{x}_i^T\boldsymbol{x}_j$
Polynomial Kernel	$(\gamma\boldsymbol{x}_i^T\boldsymbol{x}_j + r)^d$
Gaussian Kernel	$\exp(-\gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$
Sigmoid Kernel	$\tanh(\gamma\boldsymbol{x}_i^T\boldsymbol{x}_j + r)$

C. Sequential Minimal Optimization

SMO derives from the coordinate descent method, which optimizes a single variable at each step. Since there are constraints in the dual problem, we should optimize two variables at each step. Suppose that we select α_1 and α_2 to optimize and fix other variables. Then the optimization problem can be formulated as

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

where ζ is a constant. Then α_1 can be expressed as

$$\alpha_1 = \zeta y_1 - \alpha_2 y_1 y_2$$

Note that the decision function is specified as

$$f(\boldsymbol{x}) = \sum_{i=1}^{m}\alpha_i y_i\kappa(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

Let $E_i = f(\boldsymbol{x}_i) - y_i$ denote the error between the prediction and ground truth. Auxillary variables v_1 and v_2 are defined as

$$\begin{aligned}v_1 &= \sum_{i=3}^{m}\alpha_i y_i\kappa(\boldsymbol{x}_1, \boldsymbol{x}_i) \\ &= f(\boldsymbol{x}_1) - b - \alpha_1 y_1\kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) - \alpha_2 y_2\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ v_2 &= \sum_{i=3}^{m}\alpha_i y_i\kappa(\boldsymbol{x}_2, \boldsymbol{x}_i) \\ &= f(\boldsymbol{x}_2) - b - \alpha_1 y_1\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) - \alpha_2 y_2\kappa(\boldsymbol{x}_2, \boldsymbol{x}_2)\end{aligned}$$

Then the object function can be rewritten as

$$\begin{aligned}\mathcal{L}(\alpha_2) &= -\alpha_2(\zeta - \alpha_2 y_2) y_2 \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) \\ &\quad - \frac{1}{2}(\zeta - \alpha_2 y_2)^2 \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) - \frac{1}{2}\alpha_2^2 \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2) \\ &\quad - (\zeta - \alpha_2 y_2) v_1 - \alpha_2 y_2 v_2 + (1 - y_1 y_2)\alpha_2\end{aligned}$$

Take the derivative as zero, we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \alpha_2} &= -(\kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) + \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2) - 2\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2))\alpha_2 \\ &\quad + y_2 \zeta(\kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) - \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2)) \\ &\quad + (v_1 - v_2) y_2 - y_1 y_2 + y_2^2 = 0\end{aligned}$$

Let $\eta = \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) + \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2) - 2\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2)$. Note that

$$\begin{aligned}v_1 - v_2 &= f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2) + \alpha_2 y_2 \eta \\ &\quad - \zeta(\kappa(\boldsymbol{x}_1, \boldsymbol{x}_1) - \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2))\end{aligned}$$

Hence, it holds that

$$\frac{\partial \mathcal{L}}{\partial \alpha_2} = -\eta \alpha_2^* + \eta \alpha_2 + y_2(E_1 - E_2) = 0$$

which yields

$$\alpha_2^* = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}$$

There is another constraint that $0 \le \alpha_1, \alpha_2 \le C$, so we need to clip α_2^* to a feasible range. Let L and H be the lower and upper bound of α_2, we have

$$\alpha_2' = \begin{cases} L, & \alpha_2^* < L \\ \alpha_2^*, & L \le \alpha_2^* \le H \\ H, & \alpha_2^* > H \end{cases}$$

where L and H are specified as

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

$$H = \begin{cases} \min\{C, \alpha_2 + \alpha_1\}, & y_1 = y_2 \\ \min\{C, \alpha_2 - \alpha_1 + C\}, & y_1 \ne y_2 \end{cases}$$

Then α_1 will be updated as

$$\alpha_1' = \alpha_1 + y_1 y_2(\alpha_2 - \alpha_2')$$

If $0 < \alpha_i < C$, we know that $\boldsymbol{x}_i$ is a support vector. Auxillary variables b_1 and b_2 are defined as

$$\begin{aligned}b_1 &= b - E_1 - y_1 \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1)(\alpha_1' - \alpha_1) \\ &\quad - y_2 \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2)(\alpha_2' - \alpha_2) \\ b_2 &= b - E_2 - y_1 \kappa(\boldsymbol{x}_1, \boldsymbol{x}_2)(\alpha_1' - \alpha_1) \\ &\quad - y_2 \kappa(\boldsymbol{x}_2, \boldsymbol{x}_2)(\alpha_2' - \alpha_2)\end{aligned}$$

Therefore, the threshold b will be updated as

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

Now we have derived all the update rules for a single step. When training, we repeat selecting a pair (α_i, α_j) and updating them until convergence. The whole algorithm is shown in the following pseudo-code. We should note that this algorithm is a simplified version. Since the selection of (α_i, α_j) is random, it may bring severely redundant computation. Hence, it takes millions of iterations to converge.

Algorithm 1 Sequential Minimal Optimization

```
Input: Training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$, penalty parameter $C$, computation tolerance $\varepsilon$, kernel function $\kappa(\cdot,\cdot)$
Output: Multiplier $\boldsymbol{\alpha}$, threshold $b$
 1: Initialize $\boldsymbol{\alpha} \leftarrow \mathbf{0}$, $b \leftarrow 0$
 2: while not converge do
 3:     Select a pair $(\alpha_i, \alpha_j)$ randomly
 4:     Compute $E_i \leftarrow f(\boldsymbol{x}_i) - y_i$, $E_j \leftarrow f(\boldsymbol{x}_j) - y_j$
 5:     Compute $\eta = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) + \kappa(\boldsymbol{x}_j, \boldsymbol{x}_j) - 2\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$
 6:     if $\eta \le 0$ then
 7:         continue
 8:     end if
 9:     if $y_i = y_j$ then
10:         Compute $L \leftarrow \max\{0, \alpha_j + \alpha_i - C\}$
11:         Compute $H \leftarrow \min\{C, \alpha_j + \alpha_i\}$
12:     else
13:         Compute $L \leftarrow \max\{0, \alpha_j - \alpha_i\}$
14:         Compute $H \leftarrow \min\{C, \alpha_j - \alpha_i + C\}$
15:     end if
16:     Compute $\alpha_j^* \leftarrow \alpha_j + \frac{y_j(E_i - E_j)}{\eta}$
17:     Clip $\alpha_j' \leftarrow \begin{cases} L, & \alpha_j^* < L \\ \alpha_j^*, & L \le \alpha_j^* \le H \\ H, & \alpha_j^* > H \end{cases}$
18:     Compute $\alpha_i' \leftarrow \alpha_i + y_i y_j(\alpha_j' - \alpha_j)$
19:     Compute $b_1 \leftarrow b - E_i - y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)(\alpha_i' - \alpha_i) - y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)(\alpha_j' - \alpha_j)$
20:     Compute $b_2 \leftarrow b - E_j - y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)(\alpha_i' - \alpha_i) - y_j \kappa(\boldsymbol{x}_j, \boldsymbol{x}_j)(\alpha_j' - \alpha_j)$
21:     if $0 < \alpha_i' < C$ then
22:         $b' \leftarrow b_1$
23:     else if $0 < \alpha_j' < C$ then
24:         $b' \leftarrow b_2$
25:     else
26:         $b' \leftarrow \frac{1}{2}(b_1 + b_2)$
27:     end if
28:     Update $\alpha_i \leftarrow \alpha_i'$, $\alpha_j \leftarrow \alpha_j'$, $b \leftarrow b'$
29: end while
30: return multiplier $\boldsymbol{\alpha}$, threshold $b$
```

To solve this problem, we introduce a heuristic method to select (α_i, α_j). An outer loop is added to check whether the KKT conditions are satisfied. We prefer to select the α_i that violates the KKT conditions first. Then α_j is selected to maximize the step size. In practice, the choice of α_j takes too much time, so we omit it and select α_j randomly. Such heuristic method turns out to accelerate the convergence to a large extent [2].

Another technique for acceleration is cache. Repeatedly computing the kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is a waste of time. We can pre-compute the kernel matrix $\boldsymbol{K}$ before iterations, which saves a lot of time. Besides, the error vector E can be cached as well, but we skip it considering that maintaining arrays in Python is extremely slow.

D. Multi-class Strategy

Since the typical SVM is a binary classifier, we need to extend it to support multi-class classification. One-versus-one (OVO) and one-versus-rest (OVR) are two popular strategies. For a k-class classification task, OVO trains C_k^2 classifiers, where each classifier discriminate one class from another class. OVR trains k classifiers, where each classifier discriminates one class from the rest. Although OVO trains much more classifiers, each classifier will be trained on a much smaller dataset, which makes it even faster than OVR. Moreover, OVO tends to achieve higher accuracy since its dataset will be more balanced. Therefore, we apply OVO strategy to implement the Support Vector Classifier (SVC), which is consistent with the implementation of LIBSVM [5]. Details of this strategy is shown in the following pseudo-code. Since the label for SVM should be ± 1, we view the data with smaller labels as negative samples and the data with larger labels as positive samples. The prediction is decided through voting. We can expect the correct prediction to have the most votes.

Algorithm 2 One-versus-one Strategy

```
Input: Training set $\boldsymbol{X}_{\text{train}}, \boldsymbol{y}_{\text{train}}$, test data $\boldsymbol{X}_{\text{test}}$, binary classifier $\mathcal{C}$, class number $k$
Output: Test prediction $\boldsymbol{y}_{\text{predict}}$
 1: Initialize $C_k^2$ binary classifiers $\mathcal{C}_{(i,j)}$
 2: for $i = 1$ to $k$ do
 3:     for $j = i + 1$ to $k$ do
 4:         Select subset $\boldsymbol{X}_{(i,j)} \subset \boldsymbol{X}_{\text{train}}$ and $\boldsymbol{y}_{(i,j)} \subset \boldsymbol{y}_{\text{train}}$ where $\boldsymbol{y}_{(i,j)} \in \{i, j\}^m$
 5:         Train classifier $\mathcal{C}_{(i,j)}$ on subset $\boldsymbol{X}_{(i,j)}, \boldsymbol{y}_{(i,j)}$
 6:         Vote prediction $\boldsymbol{y}_{(i,j)}$ on $\boldsymbol{X}_{\text{test}}$ by $\mathcal{C}_{(i,j)}$
 7:     end for
 8: end for
 9: $\boldsymbol{y}_{\text{predict}} \leftarrow \arg\max_{1 \le i \le k} \sum_{j \ne i} \mathbb{1}[\boldsymbol{y}_{(i,j)} = i]$
10: return $\boldsymbol{y}_{\text{predict}}$
```

E. Multiple Kernel Learning

Admittedly, kernel trick is a powerful technique to extend the linear SVM to non-linear. However, the kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is usually designed artificially. In practice, we may not know which kernel function is better. Therefore, we need an adaptive method to select the kernel function, which is exactly what Multiple Kernel Learning (MKL) does.

Suppose we have l mapping functions $\phi_1, \phi_2, \ldots, \phi_l$, which correspond to l kernel functions $\kappa_1, \kappa_2, \cdots, \kappa_l$. The combined mapping function is defined as a convex combination of the l mapping functions, namely

$$\phi(\boldsymbol{x}) = \sum_{k=1}^{l} \lambda_k \phi_k(\boldsymbol{x})$$

Then the combined kernel function is specified as

$$\begin{aligned}
\kappa(\boldsymbol{x}, \boldsymbol{y}) &= \phi(\boldsymbol{x})^T \phi(\boldsymbol{y}) \\
&= \left[\sum_{k=1}^{l} \lambda_k \phi_k(\boldsymbol{x})\right]^T \left[\sum_{k=1}^{l} \lambda_k \phi_k(\boldsymbol{y})\right] \\
&= \sum_{k=1}^{l} \lambda_k^2 \phi_k(\boldsymbol{x})^T \phi_k(\boldsymbol{y}) \\
&= \sum_{k=1}^{l} \lambda_k^2 \kappa_k(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}$$

The corresponding dual problem is specified as

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

Solving $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ simultaneously is extremely difficult, so we need to use indirect optimization methods. The basic idea is to fix $\boldsymbol{\lambda}$ and solve $\boldsymbol{\alpha}$, then fix $\boldsymbol{\alpha}$ and solve $\boldsymbol{\lambda}$. Such method is called Group Lasso Minimization (GLM) [13]. The algorithm is shown in the following pseudo-code. where $\circ$ represents Hadamard product. GLM has been proved effective to find the optimal kernel weights. However, the iterative process is extremely time-consuming. Suppose GLM requires p rounds to converge, the training will be at least $l \times p$ times slower, which will be unacceptable. Therefore, we propose a heuristic method to accelerate the process. A radical method is to find

Algorithm 3 Group Lasso Minimization

```
Input: Kernel number $l$, kernel matrices $\boldsymbol{K}_1, \boldsymbol{K}_2, \ldots, \boldsymbol{K}_l$
Output: Kernel weights $\boldsymbol{\lambda}$
1: Initialize $\boldsymbol{\lambda} = \left(\frac{1}{l}, \frac{1}{l}, \ldots, \frac{1}{l}\right)^T$
2: while not converge do
3:     Compute $\boldsymbol{K} \leftarrow \sum_{k=1}^{l} \lambda_k \boldsymbol{K}_k$
4:     Solve $\boldsymbol{\alpha}$ and $b$ by SMO algorithm
5:     Compute $g_k^2 \leftarrow \lambda_k^2 (\boldsymbol{\alpha} \circ \boldsymbol{y})^T \boldsymbol{K}_k (\boldsymbol{\alpha} \circ \boldsymbol{y})$
6:     Update $\lambda_k \leftarrow \frac{g_k}{\sum_{i=1}^{l} g_i}$
7: end while
8: return kernel weights $\boldsymbol{\lambda}$
```

a closed-form solution for $\boldsymbol{\lambda}$ instead of running iterations. The Frobenius inner product of two matrices is defined as

$$\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_F = \sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{X}_{ij} \boldsymbol{Y}_{ij}$$

Then the kernel similarity is defined as

$$S(\boldsymbol{X}, \boldsymbol{Y}) = \frac{\langle \boldsymbol{X}, \boldsymbol{Y} \rangle_F}{\sqrt{\langle \boldsymbol{X}, \boldsymbol{X} \rangle_F \langle \boldsymbol{Y}, \boldsymbol{Y} \rangle_F}}$$

Hence, the kernel weights can be computed as

$$\lambda_k = \frac{S(\boldsymbol{K}_k, \boldsymbol{y}\boldsymbol{y}^T)}{\sum_{i=1}^{l} S(\boldsymbol{K}_i, \boldsymbol{y}\boldsymbol{y}^T)}$$

We should admit that this solution may not be optimal, but it controls the time within an acceptable range, which is vital for implementation and experiments.

III. Experiments

A. Convergence

Basically, we implement the SVM using SMO algorithm. To show the effectiveness of the SVM intuitively, we construct two groups of samples satisfying Gaussian distributions with different means and apply the SVM to separate them. The result of separation is shown in the following figure. According

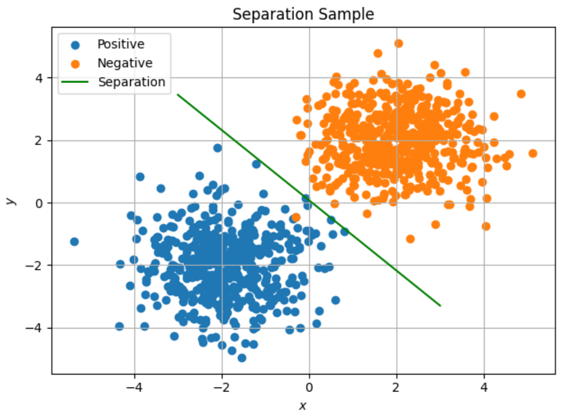

Fig. 2. Separation Sample

to the previous analysis, we implement a multi-class classifier based on SVM. To verify its convergence and effectiveness, we train the classifier on the MNIST dataset and reveal the relationship between accuracy and iterations. The convergence curve is shown in the following figure. Notice that all the parameters are set as default without tuning. We can see that as the iteration increases, the accuracy rises rapidly and converges soon, which indicates the high efficiency of the algorithm. The curve remains stable after 80 iterations. Hence, 100 iterations will be a reasonable choice for other experiments as well.

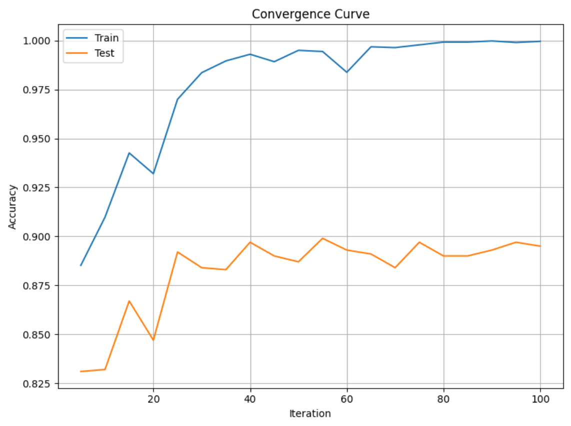

Fig. 3. Convergence Curve

B. Ablation

The next step is to figure out the influence of different parameters on the performance of the classifier. We will conduct ablation experiments on the MNIST dataset. For different kernel functions, we will research on different parameters. For the linear kernel, we will research on the penalty C and the tolerance ε. For the polynomial kernel, we will research on the scaler γ and the degree d. For the Gaussian kernel, we will research on the penalty C and the scaler γ. For the sigmoid kernel, we will research on the scaler γ and the bias r. The results are shown in the following figures. Note that all the parameters are set as default except for those we are researching on. The searching range of ε is from 10^{-3} to 10^{-6}. The searching range of d is from 1 to 5. The searching range of r is from -1 to 1. The searching range for C and γ is approximately from 2^0 to 2^{-7}, which covers almost all the reasonable values.

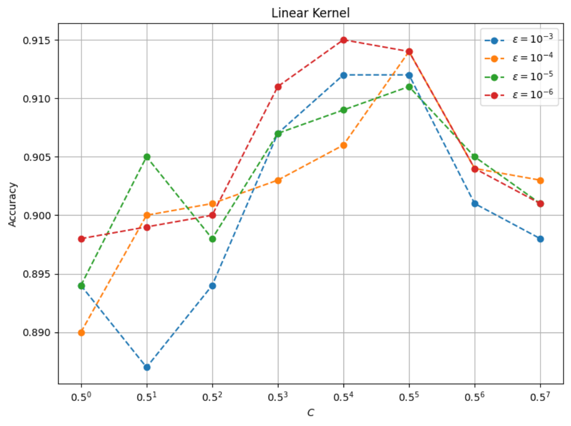

Fig. 4. Linear Ablation

Observe the results and we find multiple conclusions. Firstly, we can see that linear kernel and polynomial kernel are robust to the parameters. The accuracy remains around 90% even if the parameters are set to the extreme values. It

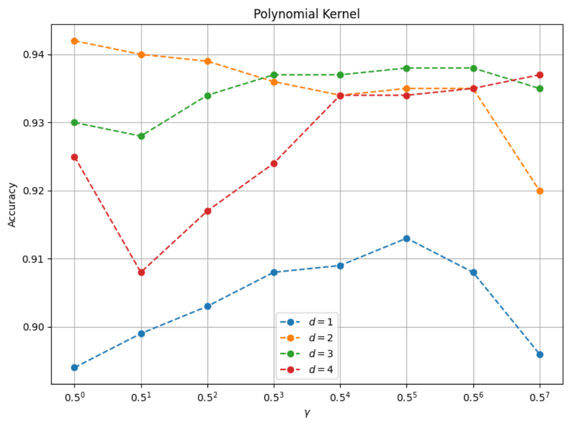

Fig. 5. Polynomial Ablation

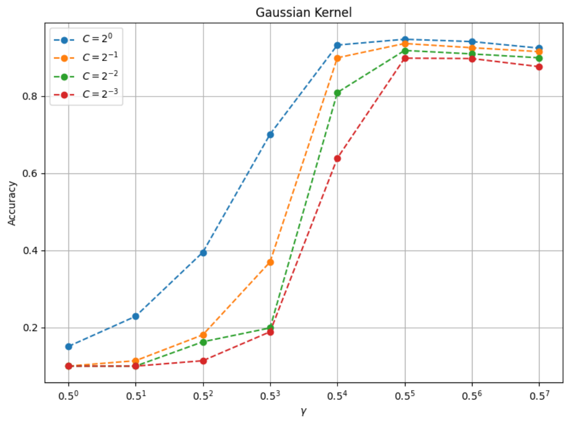

Fig. 6. Gaussian Ablation

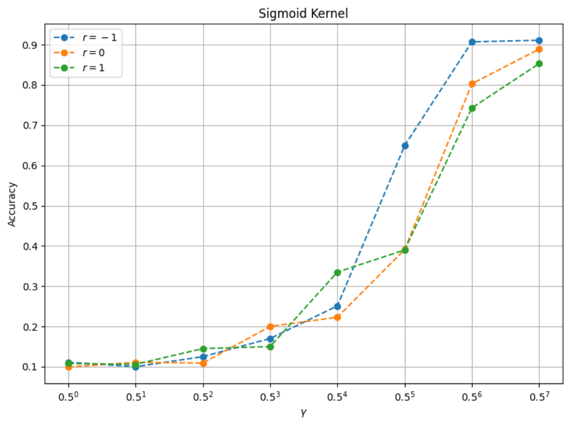

Fig. 7. Sigmoid Ablation

is not the case for Gaussian kernel and sigmoid kernel. The accuracy can be extremely low if the parameters are not set properly. But with well-tuned parameters, the Gaussian kernel can achieve the highest accuracy. Secondly, we can see that the penalty C controls the trade-off between the error rate and the generalization ability. The tolerance ε only affects the performance slightly. The polynomial kernel achieves the highest accuracy when the degree d is set to 3. The sigmoid kernel achieves the highest accuracy when the bias r is set to -1. The scaler γ can be viewed as a normalization factor for the polynomial kernel and the sigmoid kernel, but is still worth tuning. For the Gaussian kernel, the scaler γ is of vital importance, which decides the representation ability of the kernel function. When γ is too large, the classifier tends to overfit. When γ is too small, the classifier tends to underfit. Hence, a suitable γ should be tuned carefully. The penalty C also influences the performance of the Gaussian kernel, but the influence is limited compared with γ.

C. Precision

We train the classifier on the MNIST dataset and the CIFAR10 dataset. Since the size of the datasets is extremely large, we only use a part of the datasets. For each class, we reserve 500 samples in the training set and 100 samples in the test set. Hence, the training set and the test set will contain 5000 and 1000 samples respectively. Moreover, the images in the CIFAR10 dataset contain 3 channels, which makes the task quite difficult. To improve the performance, we convert the images to grayscale and extract their HOG features. To be more specific, we split each image into $n \times n$ blocks and compute the gradient of each block. Then we divide the gradient into m groups and compute the histogram of each group. The concatenated histograms are the HOG features we need. The results are shown in the following table. Note that all the parameters are well-tuned. The results

TABLE II
CLASSIFIER PRECISION

Kernel	Group	MNIST	CIFAR10	CIFAR10-HOG
Linear	Train	96.96%	76.14%	77.58%
	Test	90.70%	33.60%	39.50%
Polynomial	Train	100.00%	99.86%	100.00%
	Test	94.00%	37.70%	44.10%
Gaussian	Train	**99.68%**	99.14%	**94.54%**
	Test	**94.80%**	34.10%	**47.00%**
Sigmoid	Train	95.42%	7.96%	59.82%
	Test	92.10%	7.40%	44.70%
Combined	Train	100.00%	**99.86%**	100.00%
	Test	94.10%	**37.80%**	43.50%

show that the Gaussian kernel achieves the highest accuracy on the MNIST dataset and the CIFAR10-HOG dataset and the combined kernel achieves the highest accuracy on the CIFAR10 dataset. The polynomial kernel tends to achieve high accuracy on the training set, but falls into severe overfitting. The sigmoid kernel can avoid overfitting, but does not work in some cases. The linear kernel has the worst precision, but shows the highest efficiency. Therefore, the linear kernel and the Gaussian kernel are the most popular kernels in practice. The combined kernel is built by the heuristic MKL algorithm as the previous discussion. We can see that MKL can achieve relatively high accuracy on both the training set and the test set simultaneously. This advantage comes from the combination of multiple representations. We believe the combined kernel can defeat the single Gaussian kernel with more kernels and better weights.

D. *Efficiency*

We train the classifier with different kernels on the MNIST dataset to compare their learning and inference efficiency. The results are shown in the following table. Note that the learning time and the inference time are measured in seconds. We can see that the linear kernel and the sigmoid kernel show the highest efficiency. The polynomial kernel is also quite fast, while the Gaussian kernel is extremely slow. Since the MKL algorithm applies all the four kernels, the combined kernel shows the lowest efficiency, which is mainly caused by the Gaussian kernel. Therefore, the linear kernel will be a good choice with high demand for efficiency. If the training and inference time is not the main concern, the Gaussian kernel will be better. Compared with single kernels, MKL is more flexible. We can conveniently balance the efficiency and the precision by adjusting the kernels and their weights.

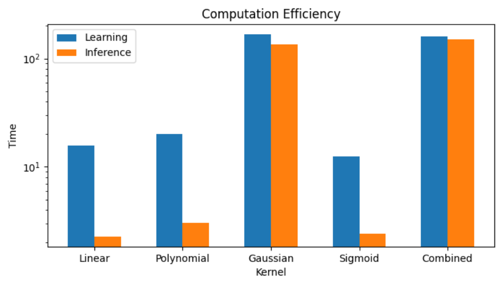

Fig. 8. Computation Efficiency

E. *Combination*

Finally, we want to study the combination of multiple kernels and reveal how the combination affects the performance of MKL. We train the classifier with different combinations of kernels on the MNIST dataset. The results are shown in the following table. Note that the L, P, G, S in the table represent the linear kernel, the polynomial kernel, the Gaussian kernel and the sigmoid kernel respectively.

TABLE III
KERNEL COMBINATION

L	P	G	S	Train	Test
✓	✓			100.00%	93.60%
✓		✓		99.90%	91.00%
✓			✓	99.86%	89.70%
	✓	✓		100.00%	93.80%
	✓		✓	100.00%	93.90%
		✓	✓	**99.46%**	**94.50%**
✓	✓	✓		100.00%	93.90%
✓	✓		✓	100.00%	93.80%
✓		✓	✓	99.88%	91.20%
	✓	✓	✓	100.00%	94.00%
✓	✓	✓	✓	100.00%	93.80%

Obeserve the results and we can see that the linear kernel lacks the ability to capture the complex patterns, which leads to a relatively low accuracy. The polynomial kernel amplifies the difference between features and achieves high accuracy on the training set, but suffers from overfitting. The combination of the Gaussian kernel and the sigmoid kernel achieve the highest accuracy on the test set, which slightly defeats the single Gaussian kernel and shows the sota performance. Beyond that, we believe that more kernels promise better interpretabilty and robustness, but their weights should be carefully designed.

IV. CONCLUSION

In this project, we derive and implement SVM and MKL from scratch and conduct experiments on the MNIST dataset and the CIFAR10 dataset. We study the influence of different parameters and different kernels on the performance of the classifier. We find that both the penalty C and the scaler γ control underfitting and overfitting, which are vital to the performance of the classifier. We conclude that the linear kernel has the highest efficiency and the Gaussian kernel has the highest accuracy, which make them the most popular kernels in practice. We propose the heuristic MKL algorithm based on GLP, which greatly improves the efficiency of MKL. We find that the combination of the Gaussian kernel and the sigmoid kernel shows the sota performance. Until today, the improvement of MKL is still an open problem in the field of machine learning. Optimization algorithms with higher efficiency and better performance are more than promising.

REFERENCES

[1] Zhou Z H. Machine learning[M]. Springer Nature, 2021.
[2] Harrington P. Machine learning in action[M]. Simon and Schuster, 2012.
[3] Hsu C W, Chang C C, Lin C J. A practical guide to support vector classification[J]. 2003.
[4] Cortes C, Vapnik V. Support-vector networks[J]. Machine learning, 1995, 20: 273-297.
[5] Chang C C, Lin C J. LIBSVM: a library for support vector machines[J]. ACM transactions on intelligent systems and technology (TIST), 2011, 2(3): 1-27.
[6] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine learning in Python[J]. the Journal of machine Learning research, 2011, 12: 2825-2830.
[7] Platt J. Sequential minimal optimization: A fast algorithm for training support vector machines[J]. 1998.
[8] Lanckriet G R G, Cristianini N, Bartlett P, et al. Learning the kernel matrix with semidefinite programming[J]. Journal of Machine learning research, 2004, 5(Jan): 27-72.
[9] Price S R, Anderson D T, Havens T C, et al. Kernel Matrix-Based Heuristic Multiple Kernel Learning[J]. Mathematics, 2022, 10(12): 2026.
[10] Sonnenburg S, Rätsch G, Schäfer C, et al. Large scale multiple kernel learning[J]. The Journal of Machine Learning Research, 2006, 7: 1531-1565.
[11] Bach F R, Lanckriet G R G, Jordan M I. Multiple kernel learning, conic duality, and the SMO algorithm[C]. Proceedings of the twenty-first international conference on Machine learning. 2004: 6.
[12] Rakotomamonjy A, Bach F, Canu S, et al. SimpleMKL[J]. Journal of Machine Learning Research, 2008, 9: 2491-2521.
[13] Xu Z, Jin R, Yang H, et al. Simple and efficient multiple kernel learning by group lasso[C]. Proceedings of the 27th international conference on machine learning (ICML-10). 2010: 1175-1182.