Report of Class Project: Implementation of SVM

MoeKid101

1. Introduction

In this class project I will implement the *support vector machine* (SVM) by myself.

2. Theory

In this section, several critical equations (based on Platt's paper) about SVM as well as my understandings are provided. To help myself understand all the mathematics behind SVM, a brief mathematical derivation is provided in the appendix.

2.1. SVM Routine

Given training set $\{(\boldsymbol{x}^{(i)}, y^{(i)}) \,|i = 1, 2, ..., n\}$ where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is a d-dimension sample and $y^{(i)} \in \{-1, +1\}$ is the label. Our problem is the following optimization problem:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t. } \forall i, y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \geq 1 - \xi_i \\ &\quad \text{and } \boldsymbol{\xi} \geq \mathbf{0} \end{aligned} \tag{1}$$

whose Lagrange duality problem is

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y^{(i)}y^{(j)}\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\rangle\right) \\ &\text{s.t. } \mathbf{0} \leq \boldsymbol{\alpha} \leq C\mathbf{1} \text{ and } \sum_{i=1}^{n}\alpha_i y^{(i)} = 0 \end{aligned} \tag{2}$$

To solve the problem Eq.(2), *sequential minimal optimization* (SMO) is introduced. In SMO, $n-2$ variables are fixed and we solve it for the rest two. WLOG, suppose the rest two variables are α_1 and α_2, then the updates are

$$\begin{aligned} \alpha_2^{\text{new,clipped}} &= \begin{cases} H, & \alpha_2^{\text{new}} \geq H \\ \alpha_2^{\text{new}}, & L < \alpha_2^{\text{new}} < H \\ L, & \alpha_2^{\text{new}} \leq L \end{cases} \\ \alpha_1^{\text{new}} &= \alpha_1 - y^{(1)}y^{(2)}\left(\alpha_2^{\text{new,clipped}} - \alpha_2^{\text{new}}\right) \end{aligned} \tag{3}$$

where

$$\begin{aligned} L &= \begin{cases} \max\{0, \alpha_1 + \alpha_2 - C\}, & y^{(1)}y^{(2)} = 1 \\ \max\{0, \alpha_2 - \alpha_1\}, & y^{(1)}y^{(2)} = -1 \end{cases} \\ H &= \begin{cases} \min\{C, \alpha_1 + \alpha_2\}, & y^{(1)}y^{(2)} = 1 \\ \min\{C, C + \alpha_2 - \alpha_1\}, & y^{(1)}y^{(2)} = -1 \end{cases} \end{aligned} \tag{4}$$

In the implementation, SMO stops when the KKT conditions (Eq.(7)) are satisfied.

$$\exists b \text{ s.t. } \begin{cases} y^{(i)}h(\boldsymbol{x}^{(i)}) \geq 1, & \text{if } \alpha_i = 0 \\ y^{(i)}h(\boldsymbol{x}^{(i)}) = 1, & \text{if } 0 < \alpha_i < C \\ y^{(i)}h(\boldsymbol{x}^{(i)}) \leq 1, & \text{if } \alpha_i = C \end{cases} \tag{5}$$

where $h(\boldsymbol{x}^{(i)}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ is the output of SVM model. Therefore, it's necessary to keep record of b and update it after every step. Here b is updated under the principle of making $(\boldsymbol{x}^{(1)}, y^{(1)})$ and $(\boldsymbol{x}^{(2)}, y^{(2)})$ satisfy Eq.(7). It is defined that

$$\begin{aligned} b_1^{\text{new}} = &- e_1 - y^{(1)}\left\|\boldsymbol{x}^{(1)}\right\|^2(\alpha_1^{\text{new}} - \alpha_1) \\ &- y^{(2)}\left\langle \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\right\rangle\left(\alpha_2^{\text{new,clipped}} - \alpha_2\right) + b \end{aligned}$$

as well as a symmetric definition of b_2^{new}. Platt concluded that

$$b^{\text{new}} = \begin{cases} b_1^{\text{new}}, & \text{if } \alpha_1^{\text{new}} \in (0, C) \\ b_2^{\text{new}}, & \text{if } \alpha_2^{\text{new}} \in (0, C) \\ \frac{1}{2}\left(b_1^{\text{new}} + b_2^{\text{new}}\right), & \text{otherwise and } L \neq H \\ \text{skip}, & \text{otherwise and } L = H \end{cases} \tag{6}$$

Finally, the parameters to update are chosen according to the following heuristics:

- The non-bound samples (i.e. $\alpha \in (0, C)$) are more likely to violate KKT conditions, so we always select non-bound sample to be α_1 as long as there exists one.
- If there is no non-bound sample, then the whole dataset is scanned through to check for samples violating KKT conditions.
- When α_1 is selected, α_2 is chosen to be the one (approximately) maximizing $|e_1 - e_2|$.

2.2. My Interpretation

Section 2.1 describes a clear routine of training SVMs using SMO. But there remains some questions. Firstly, Eq.(7) took the threshold b as the Lagrange multiplier, the reason of which is not clarified by Platt. Secondly, Platt gave a paragraph describing Eq.(6) but didn't cover the reasons, and the threshold update being skipped given $L = H$ was only given in the pseudo-code.

It took me some time to figure out the answers to these two questions, and my understandings are presented below.

First, SMO can stop when the KKT conditions (Eq.(7)) of the dual problem (Eq.(2)) are satisfied.

$$\begin{cases} y^{(i)}h(\boldsymbol{x}^{(i)}) \geq 1, & \text{if } \alpha_i = 0 \\ y^{(i)}h(\boldsymbol{x}^{(i)}) = 1, & \text{if } 0 < \alpha_i < C \\ y^{(i)}h(\boldsymbol{x}^{(i)}) \leq 1, & \text{if } \alpha_i = C \end{cases} \tag{7}$$

where $h(\boldsymbol{x}^{(i)}) = \boldsymbol{w}^\top \boldsymbol{x} + \lambda$ and λ is the Lagrange multiplier corresponding to constraint $\sum_i \alpha_i y^{(i)} = 0$. However, we notice that Eq.(2) is exactly a part of the KKT conditions of the primal problem Eq.(1) as long as λ is substituted by b. Therefore, Platt et al. directly used $h(\boldsymbol{x}^{(i)}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ as the output of SVM.

The substitution $\lambda \to b$ eliminates the need for searching λ to verify "there exists λ" in KKT theorem, at the cost of maintaining the threshold b. b is updated according to the principle that $(\boldsymbol{x}^{(1)}, y^{(1)})$ and $(\boldsymbol{x}^{(2)}, y^{(2)})$ satisfy the KKT conditions.

Case I If $\alpha_1^{\text{new}}, \alpha_2^{\text{new}} \in (0, C)$, then to make $(\boldsymbol{x}^{(1)}, y^{(1)})$ satisfy KKT condition,

$$\begin{aligned} b_1^{\text{new}} =& y^{(1)} - \boldsymbol{w}^{\text{new}\top}\boldsymbol{x}^{(1)} \\ =& e_1 - y^{(1)}\left\|\boldsymbol{x}^{(1)}\right\|^2(\alpha_1^{\text{new}} - \alpha_1) \\ & - y^{(2)}\left\langle \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)} \right\rangle \left(\alpha_2^{\text{new,clipped}} - \alpha_2\right) + b \end{aligned}$$

and a similar equation holds for b_2^{new}. In this case, $b_1^{\text{new}} = b_2^{\text{new}}$ because

$$\begin{aligned} b_1^{\text{new}} - b_2^{\text{new}} =& (e_1 - e_2) - y^{(1)}\left\langle \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)} \right\rangle \Delta\alpha_1 \\ & - y^{(2)}\left\langle \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)} \right\rangle \Delta\alpha_2 \\ =& (e_1 - e_2) + y^{(2)}\left\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)}\right\|^2 \Delta\alpha_2 = 0 \end{aligned}$$

(the second equality holds because $\Delta_1 = -y^{(1)}y^{(2)}\Delta_2$)

Case II If $\alpha_1^{\text{new}} \in (0, C)$ and $\alpha_2^{\text{new}} \in \{0, C\}$, then b_1^{new} is still given by the same equation. Since we have $y^{(2)}\left(\boldsymbol{w}^{\text{new}\top}\boldsymbol{x}^{(2)} + b_1^{\text{new}}\right) = 1$ according to the first case, b_1^{new} is a valid value whether $\alpha_2 = 0$ or $\alpha_2 = C$. Consequently, we can directly set $b^{\text{new}} = b_1^{\text{new}}$ to obtain a valid threshold.

Case III If $\alpha_1^{\text{new}}, \alpha_2^{\text{new}} \in \{0, C\}$, we need further classification.

Case III.1 When $\alpha_1^{\text{new}} = \alpha_2^{\text{new}} = 0$ and $y^{(1)}y^{(2)} = -1$. In this case we have $y^{(1)}h(\boldsymbol{x}^{(1)}) \geq 1$ and $y^{(2)}h(\boldsymbol{x}^{(2)}) \geq 1$. Because $y^{(1)}$ and $y^{(2)}$ are of different signs, the constraint is, in fact, an interval with bounds b_1^{new} and b_2^{new}. Platt chose the update to be $b^{\text{new}} = \frac{1}{2}\left(b_1^{\text{new}} + b_2^{\text{new}}\right)$.

The conclusion also holds for: (1) $\alpha_1^{\text{new}} = \alpha_2^{\text{new}} = C$ and $y^{(1)}y^{(2)} = -1$, (2) $\alpha_1^{\text{new}} + \alpha_2^{\text{new}} = C$ and $y^{(1)}y^{(2)} = 1$.

Case III.2 Consider the case when $\alpha_1^{\text{new}} = \alpha_2^{\text{new}} = 0$ and $y^{(1)}y^{(2)} = 1$. Using $L = \max\{0, \alpha_1 + \alpha_2 - C\}$ and $H = \min\{C, \alpha_1 + \alpha_2\}$ and $\alpha_1^{\text{new}} + \alpha_2^{\text{new}} = \alpha_1 + \alpha_2$ we have $L = H = 0$.

The conclusion $L = H$ also holds for: (1) $\alpha_1^{\text{new}} = \alpha_2^{\text{new}} = C$ and $y^{(1)}y^{(2)} = 1$, (2) $\alpha_1^{\text{new}} + \alpha_2^{\text{new}} = C$ and $y^{(1)}y^{(2)} = -1$.

In Platt's paper, the pseudo-code directly skipped the case where $L = H$, because here α_1 and α_2 gets no update at all.

3. Single-Kernel SVM

3.1. Kernels

In this project, *linear kernel*, *radial basis function kernel* (RBF), and *polynomial kernel* are considered.

$$K_{\text{lin}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \boldsymbol{x}^{(i)\top}\boldsymbol{x}^{(j)} \tag{8}$$

$$K_{\text{rbf}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\gamma\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2\right) \tag{9}$$

$$K_{\text{poly}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \left(\boldsymbol{x}^{(i)\top}\boldsymbol{x}^{(j)} + c\right)^d \tag{10}$$

3.2. Binary SVM

To build a first basic binary SVM classifier, I resorted to Platt's pseudo-code given in his paper. In Platt's pseudo-code, the kernel function $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$ is required very often and repeatedly, so I compute and store the kernel matrix before training the SVM. Such attempt works because, in our project, the number of training samples are limited to 500 each classes, and storing 500×500 matrix is acceptable. Obviously larger-scale SVMs require different implementation.

$*$ **Accelerate RBF Kernel**: To calculate RBF kernel matrix using Eq.(9), we use equation $\left\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\right\|^2 = \left\|\boldsymbol{x}^{(i)}\right\|^2 + \left\|\boldsymbol{x}^{(j)}\right\|^2 - 2\boldsymbol{x}^{(i)\top}\boldsymbol{x}^{(j)}$. The first two terms can be seen as the extension of vector $\boldsymbol{m} = \left[\left\|\boldsymbol{x}^{(1)}\right\|^2, \left\|\boldsymbol{x}^{(1)}\right\|^2, ..., \left\|\boldsymbol{x}^{(n)}\right\|^2\right]^\top$ along different axes, so `broadcast` in `numpy` can be utilized to speed up calculation; the third term is exactly the linear kernel matrix, which requires only matrix multiplication.

In trying to implement the SMO directly following Platt's pseudo-code, I found a problem for large training

tasks that: the algorithm is almost unable to converge, but early stopping generates no bad result.

For example, taking images labelled 1 and 2 in MNIST to train the SVM for 100 iterations (didn't converge), some statistics are shown in Fig.1. We see from the figure that each iteration witnesses more than 100 pair of α_i, α_j changed, which means we are far from convergence. However, the error rate has maintained a very low level since about 10 iterations, and $\|\boldsymbol{w}^{\text{new}} - \boldsymbol{w}^{\text{new}}\|_1$ as well as $\|\boldsymbol{\alpha}^{\text{new}} - \boldsymbol{\alpha}^{\text{new}}\|_1$ quickly drops to a low level so that there are only minor changes to our model.

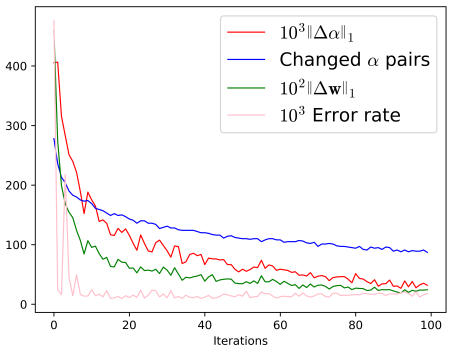


Figure 1. Binary classification statistics.

The reason for such problem is that Platt's pseudo-code ends only if the KKT conditions are satisfied by every sample. However, more often we don't have to stick to the KKT conditions so strictly to get a reliable classification model, especially in the context of image classification. Therefore all our experiments end at a maximum of 30 iterations unless otherwise stated.

3.3. Multi-Class SVM

Having obtained a binary SVM classifier, we extend it into multi-class SVM to test its performance on MNIST and CIFAR-10. Popular ways of constructing multi-class classifiers from binary ones include: *one-against-one* (OAO) models and *one-against-rest* (OAR) models. We experiment different combinations of parameters.

Observation: Recall Eq.(2) and notice that multi-kernel SVM requires weighted average of different kernels, we give an equivalent problem description in Eq.(11) which tells us that multiplying a constant M to the kernel matrix is equivalent to multiplying M to C.

$$\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^{n} \alpha_i - \frac{M}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \rangle \right)$$
$$\text{s.t. } \mathbf{0} \le \boldsymbol{\alpha} \le \frac{C}{M}\mathbf{1} \text{ and } \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \qquad (11)$$

Therefore, in our task, we can determine a fixed C in the first place and test different values of M multiplied to the kernel matrix as an equivalent to test $C' = C/M$. Through such operation we can gain some insights into the parameter space in multi-kernel cases and reduce the workload there. Now we describe our experiments below.

3.3.1 Parameter Selection (MNIST)

For `linear` kernel, there is no parameter for the kernel matrix itself, so we are only searching for an optimal M. As is shown in Fig.2, the best model is achieved at $M = 0.03$ using OAO method (equivalent of $C \approx 33$) with an error rate of 7.6%. It is also evident that compared to OAR models, OAO models are much more robust to selection of parameters.

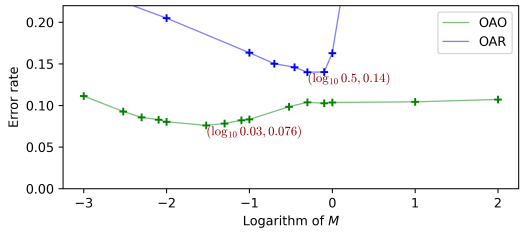


Figure 2. Linear kernel performance under different Ms.

For `rbf` kernel, we search for both γ and M. First we show the performance of models under different γ values in Fig.3. Whichever method we choose (OAO or OAR), the best model is achieved at $\gamma = 0.04$ and the error rate is as low as 3.94%. Meanwhile, when searching for a good M, I found that the performance of models remains almost the same for different Ms in a very large range of $[0.01, 100]$. Therefore we take M as simple as 1 for our models.

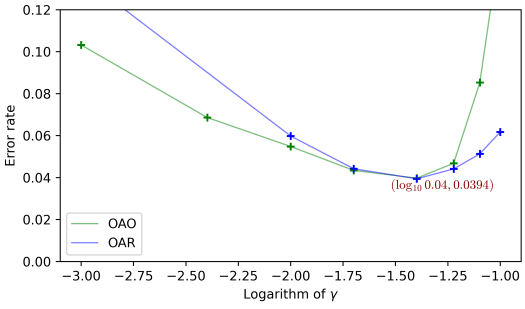
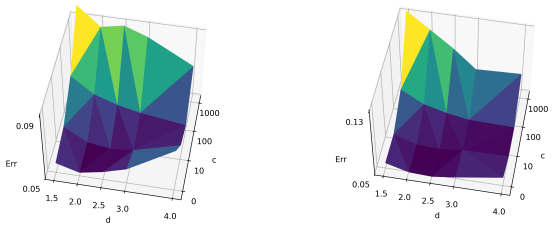


Figure 3. RBF kernel performance under different γs.

For `poly` kernel, we search for c, d and M together. We iterate through combinations of $c = 0, 10, 100, 1000$ and $d = 1.5, 2.0, 2.5, 3.0, 4.0$ and got the results shown in Fig.4. Although OAR is still more sensitive to changes in parameters, both methods yields similar optimal results: $c = 10$,

$d = 2.5$ achieving 4.30% error rate for OAR and 4.95% error rate for OAO. Similar to the `rbf` kernel case, I found that changing M doesn't take much effect here, so I took $M = 1$ too.



(a) OAO (b) OAR

Figure 4. Polynomial kernel performance under different c, ds.

Now it is concluded that the optimal parameters (together with the performance contrast with `sklearn` benmarks) for each of the three kernels are as shown in Table.1. In the table $C = 1/M$ is given to eliminate ambiguity, and since `sklearn` doesn't allow non-integer degrees as input, we take $d = 3$ instead as the corresponding benchmark. From the table we know that we have almost achieved our best in classfying MNIST images without any prior knowledge of the dataset. The best model achieved (`rbf`) is tested in more detail and the confusion matrix (which gives all the information we need) is shown in Fig.5.

Table 1. Performance of Best Models

Kernel (Method)	Parameters	Error rate (benchmark)
lin (OAO)	$C = 33$	**0.0760** (0.0911)
rbf (OAO)	$\gamma = 0.04$, $C = 1$	**0.0394** (0.0398)
poly (OAR)	$c = 10$, $d = 2.5$, $C = 1$	**0.0430** (0.0697)

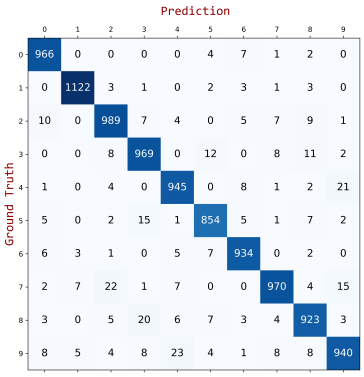


Figure 5. The confusion matrix for best `rbf` kernel model ($\gamma = 0.04$.)

Our model gets better performance on MNIST than `sklearn` baseline, at the cost of long training times and long validation time. Given a 5000-sample training set with 10 classes, `sklearn` finishes training as well as testing on my laptop within about 3sec, but it took as long as half a minute for my SMO algorithm to train and test a model.

3.3.2 Parameter Selection (CIFAR-10)

Compared to MNIST, classifying images in CIFAR-10 is a much harder task. Except from two extra channels, we suffer from large intra-class variances. It is in essence a task to be solved by convolutional neural networks, so we won't expect good results on CIFAR-10. Although we can achieve some progress using techniques including feature extraction, but that is beyond our consideration. We focus instead on the SVM algorithm itself.

Exactly the same exeriments (as those for MNIST) are carried out for the data on CIFAR-10, with results show below in Fig.6, Fig.7, Fig.8, Fig.9, and Table.2.

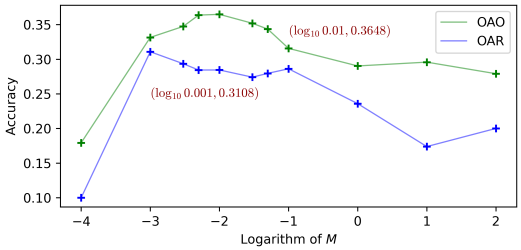


Figure 6. Linear kernel performance under different Ms.

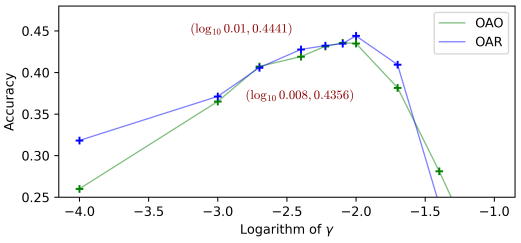
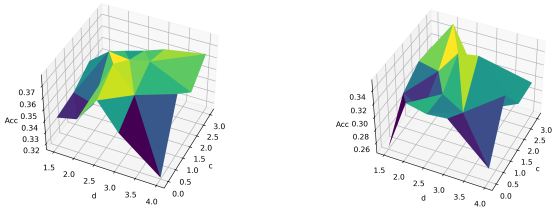


Figure 7. RBF kernel performance under different γs.



(a) OAO (b) OAR

Figure 8. Polynomial kernel performance under different c, ds.

Table 2. Performance of Best Models

Kernel (Method)	Parameters	Accuracy (benchmark)
lin (OAO)	$C = 100$	**0.3648** (0.2954)
rbf (OAR)	$\gamma = 0.01, C = 1$	**0.4441** (0.4391)
poly (OAO)	$c = 10, d = 2.5, C = 1$	0.3776 (**0.3840**)

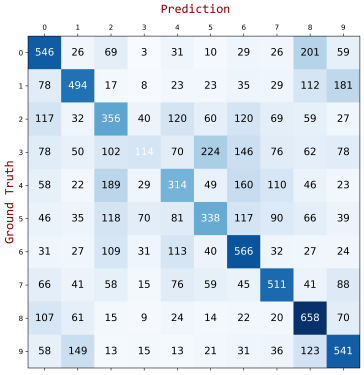


Figure 9. The confusion matrix of best `rbf` kernel model.

I found it strange that it took `sklearn` almost 90sec to perform training and then test its model in this case since the number of samples isn't increased compared to MNIST case. Maybe `sklearn` uses totally different method to implement SVM? I'm not sure.

4. Multiple Kernel Learning

It is a natural idea that, given several working kernels, if we are still not getting desired results, we might combine these kernels and it is expected that some progress is obtained. Mathematically, assuming we have m kernels $K_j(\cdot,\cdot)$ (where $j = 1, 2, ..., m$), some popular ways of combination include *linear combination* (weighted average) and *mutliplication*.

$$K_{\text{avg}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \sum_{a=1}^{m} \eta_a K_a(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \tag{12}$$

$$K_{\text{mtp}}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \prod_{a=1}^{m} K_a(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})^{\eta_a} \tag{13}$$

Obviously, the definition of *semi-definiteness* ensures us that $K_{\text{avg}}(\cdot,\cdot)$ generates valid kernels; and the Schur product theorem tells us that $K_{\text{mtp}}(\cdot,\cdot)$ is also valid.

In this case searching for a real combination of "optimal" parameters is almost impossible. I have to apologize for my empirical judgements here, but I will still explain my choices briefly.

4.1. Parameter for MNIST

(Here OAO but not OAR is used.)

I was starting with the *linear combination* way at first. I began with the combination of `poly` kernel and `rbf` kernel because I have found in the experiments in the last section that these two kernels maintain almost the same performance in a large range of M (which means multiplying η_{rbf} and η_{poly} with a constant together won't change the results) and they are the ones performing well. Having obtained the following results, I was convinced that I had found the optimal ratio of $\eta_{rbf}/\eta_{poly} \approx 10000$.

$\eta_{rbf} = 10, \eta_{poly} = 0.001$	**0.0441**
$\eta_{rbf} = 5, \eta_{poly} = 0.001$	0.0507
$\eta_{rbf} = 20, \eta_{poly} = 0.001$	0.0514

Then I tried different values of η_{lin} added to such results, all leading to loss of accuracy. Consequently, it was very likely that $[\eta_{lin}, \eta_{rbf}, \eta_{poly}] = [0, 10, 0.001]$ forms a stationary point. It seemed that everything was telling me I had obtained the optimal performance for *linear combination*.

Then I resorted to products of kernels. In this case the extra parameters are directly replaceable by d and γ, so my efforts lay in tuning d, c and γ.

I found that the performance gets better with smaller d, and accuracy remained almost the same when $d < 1$. Therefore, I took the `linear` kernel to multiply with `rbf` kernel instead. Finally, tuning γ as the only parameter became a simple task. The optimal error rate was 3.51% with $\gamma = 0.022$. The confusion matrix generated by this model is shown in Fig.10.

$K_{\text{poly},10,2.5} \odot K_{\text{rbf},0.04}$	0.0437
$K_{\text{poly},10,2.5} \odot K_{\text{rbf},0.02}$	0.0397
$K_{\text{poly},10,1.5} \odot K_{\text{rbf},0.02}$	0.0372
$K_{\text{lin}} \odot K_{\text{rbf},0.02}$	0.0361
$K_{\text{lin}} \odot K_{\text{rbf},0.022}$	**0.0351**

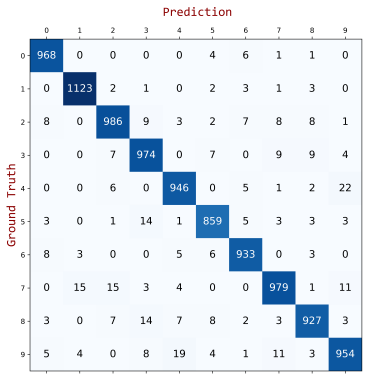


Figure 10. The confusion matrix of best multi-kernel model.

4.2. Parameter for CIFAR

Again I wasn't getting smooth here. I failed at multiplying kernels, and linear combinations didn't bring about considerable enhancement in accuracy. The best parameter I found was $[\eta_{lin}, \eta_{poly}, \eta_{rbf}] = [0, 2 \times 10^{-6}, 200]$ corresponding to an accuracy of 45.32% (confusion matrix shown in Fig.11). Taking the overall low performance into account, the 0.9% raise in accuracy is an almost totally insignificant change. However, limited by time I got, I could not conduct more experiments to test better parameters.

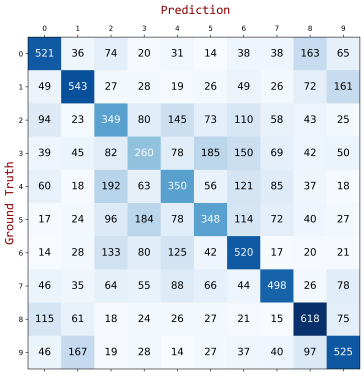


Figure 11. Confusion matrix of best multi-kernel model.

5. Conclusion and Summary

In this report I have dived into the mathematical principles and implementation of *support vector machines*. In fact I would say it is an algorithm harder to implement than to understand. Most of my time were wasted debugging the algorithm, only to find it was minor mistakes that made the subtle algorithm fail.

Whatever hardships I have gone through, I must say it is a great ecstasy to implement a well-known machine learning algorithm and even outrun `sklearn` implementation on MNIST and CIFAR-10. And of course there remains something to be improved.

It is noticeable that experimenting on CIFAR-10 (one OAO model about 40s) is more time-consuming than it is on MNIST (8s). And meanwhile, I found that during either training or testing, only less than 20% of the CPU capacity was utilized. The former problem is mainly caused by increased cost to calculate kernel matrix; while the latter is because `python` doesn't incorporate `multiprocessing` by default. Therefore, ideally we might calculate kernel matrix with GPU and introduce multi-kernel computation to boost the SMO algorithm.

What's more, in fact, I'm still wondering why my algorithm just didn't converge despite its relatively satisfactory performance achieved. Although in the context of image classification, strict convergence is never an emergent problem, I still want to know how exactly is the solution given by my algorithm different from that given by a mature implementation.

To sum up, I really enjoyed the days fighting with SMO and SVM. Thank you for taking the time to read my report.

A. Mathematical Principles

In the context of SVM, we are solving binary classification. Mathematically, our training set is assumed to be $\{(\boldsymbol{x}^{(i)}, y^{(i)}) | i = 1, 2, ..., n\}$ where $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is a d-dimension sample and $y^{(i)} \in \{-1, +1\}$ is the label. Our problem is defined by maximizing the geometric margin whose definition is given below.

Definition Geometric Margin: Given a dataset $\mathcal{D}$, if we separate it with line $h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, then the geometric margin with respect to the dataset is $\gamma = \min_{(\boldsymbol{x},y)\in\mathcal{D}} \gamma(\boldsymbol{x}, y)$ where $\gamma(\boldsymbol{x}, y) = y\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) / \|\boldsymbol{w}\|$.

Then the target of SVM is:

$$\begin{aligned}
&\max_{\gamma,\boldsymbol{w},b} \gamma \text{ s.t. } \forall i, y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) / \|\boldsymbol{w}\| \geq \gamma \\
\Leftrightarrow &\max_{\gamma,\boldsymbol{w},b} \gamma \text{ s.t. } \forall i, y^{(i)}\left(\left(\frac{\boldsymbol{w}}{\gamma}\right)^\top \boldsymbol{x}^{(i)} + \frac{b}{\gamma}\right) \geq 1 \\
&\text{and } \|\boldsymbol{w}\| = 1 \\
\Leftrightarrow &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \text{ s.t. } \forall i, y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \geq 1 \qquad (14)
\end{aligned}$$

If we introduce soft margins, allowing some samples to violate the condition $y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x} + b\right) \geq 1$, then the optimization problem is

$$\begin{aligned}
&\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^n \xi_i \\
&\text{s.t. } \forall i, y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) \geq 1 - \xi_i \\
&\text{and } \boldsymbol{\xi} \geq \boldsymbol{0} \qquad (15)
\end{aligned}$$

A.1. Lagrange Duality

Notice that all the constraints are affine functions of variable $\boldsymbol{w}, b, \boldsymbol{\xi}$, and we can simply take $\boldsymbol{w} = \boldsymbol{0}$ and $\boldsymbol{\xi}$ large enough to obtain an inner point. Therefore, the Slater's conditions holds naturally. Using the strong duality, we have

$$\begin{aligned}
(15) \Leftrightarrow \max_{\boldsymbol{\alpha}\geq\boldsymbol{0},\boldsymbol{r}\geq\boldsymbol{0}} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} &\left(\frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^n \xi_i - \boldsymbol{r}^\top\boldsymbol{\xi}\right. \\
&\left. - \sum_{i=1}^n \alpha_i\left(y^{(i)}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b\right) - 1 + \xi_i\right)\right)
\end{aligned}$$

For fixed $\boldsymbol{\alpha}$ and $\boldsymbol{r}$, we can take the derivative of the objective function f and set it to zero so as to get a minimum with respect to $\boldsymbol{w}$ and $\boldsymbol{\xi}$.

$$\begin{aligned}
\nabla_{\boldsymbol{w}} f &= \boldsymbol{w} - \sum_{i=1}^n \alpha_i y^{(i)} \boldsymbol{x}^{(i)} = 0 \\
\nabla_b f &= \sum_{i=1}^n \alpha_i y^{(i)} = 0 \\
\nabla_{\boldsymbol{\xi}} f &= C\boldsymbol{1} - \boldsymbol{\alpha} - \boldsymbol{r} = 0
\end{aligned}$$

Substituting the parameters, the resulting optimization problem is

$$\begin{aligned}
&\max_{\boldsymbol{\alpha}} \left(\sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha_i\alpha_j y^{(i)}y^{(j)}\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\rangle\right) \\
&\text{s.t. } \boldsymbol{0} \leq \boldsymbol{\alpha} \leq C\boldsymbol{1} \text{ and } \sum_{i=1}^n \alpha_i y^{(i)} = 0 \qquad (16)
\end{aligned}$$

With Eq.(16) we can introduce the kernel method which is simply replacing the inner product $\langle \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\rangle$ with any other well-defined kernel $K\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}\right)$.

A.2. Sequential Minimal Optimization

The remaining problem is to solve the optimization problem Eq.(16). The most popular algorithm is *sequential minimal optimization* (SMO). In SMO, $n-2$ variables are fixed and we solve it for the rest two. WLOG, suppose the rest two variables are α_1 and α_2, and the problem is

$$\begin{aligned}
\max_{\alpha_1,\alpha_2} &\alpha_1 + \alpha_2 - \frac{1}{2}\left\|\boldsymbol{x}^{(1)}\right\|^2\alpha_1^2 - \frac{1}{2}\left\|\boldsymbol{x}^{(2)}\right\|^2\alpha_2^2 \\
&- \alpha_1\alpha_2 y^{(1)}y^{(2)}\langle \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}\rangle \\
&- \left\langle \sum_{j\neq 1,2} \alpha_j y^{(j)}\boldsymbol{x}^{(j)}, \alpha_1 y^{(1)}\boldsymbol{x}^{(1)} - \alpha_2 y^{(2)}\boldsymbol{x}^{(2)}\right\rangle \\
\text{s.t. } &0 \leq \alpha_1, \alpha_2 \leq C, \alpha_1 y^{(1)} + \alpha_2 y^{(2)} = k
\end{aligned}$$

where $k = -\sum_{i\neq 1,2} \alpha_i y^{(i)}$ is a constant.

First we ignore the constraints $0 \leq \alpha_1, \alpha_2 \leq C$ and eliminate α_1 using $\alpha_1 = y^{(1)}k - y^{(1)}y^{(2)}\alpha_2$. The resulting objective function is a quadratic function of α_2, whose quadratic coefficient A and linear coefficient B are given by

$$\begin{aligned}
A &= -\frac{1}{2}\left\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)}\right\|^2 \\
B &= y^{(2)}\left(e_1 - e_2\right) + \alpha_2\left\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)}\right\|^2
\end{aligned}$$

where $e_i = \boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b - y^{(i)}$ is the prediction error on the i-th sample. Therefore, the minimizing parameter is

$$\alpha_2^{\text{new}} = -\frac{B}{2A} = \alpha_2 + \frac{y^{(2)}(e_1 - e_2)}{\left\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)}\right\|^2}$$

Next, the constraints $0 \leq \alpha_1, \alpha_2 \leq C$ is taken into account. If $y^{(1)}y^{(2)} = 1$, then the lower bound L and upper bound H are

$$L = \max\{0, \alpha_1 + \alpha_2 - C\}, H = \min\{C, \alpha_1 + \alpha_2\}$$

If $y^{(1)}y^{(2)} = -1$, then they are

$$L = \max\{0, \alpha_2 - \alpha_1\}, H = \min\{C, C + \alpha_2 - \alpha_1\}$$

Then the final update formula is given by

$$\alpha_2^{\text{new,clipped}} = \begin{cases} H, & \alpha_2^{\text{new}} \geq H \\ \alpha_2^{\text{new}}, & L < \alpha_2^{\text{new}} < H \\ L, & \alpha_2^{\text{new}} \leq L \end{cases}$$

$$\alpha_1^{\text{new}} = \alpha_1 - y^{(1)} y^{(2)} \left(\alpha_2^{\text{new,clipped}} - \alpha_2^{\text{new}} \right) \tag{17}$$