Foundational Development of Kernelized Support Vector Machines

Mouhssine Rifaki
mouhssine.rifaki@psl.eu

1 Introduction

In this project, I focused on training classifiers from scratch on high-dimensional datasets (MNIST in my case) using kernel-based methods. My work specifically focused on exploring and evaluating variants of Support Vector Machines.

2 Outline

The project itself had three main phases, each with its own challenges. Phase one focused on preprocessing the training samples. Tasks like noise filtering, scaling, and tilt correction were all tackled here to try to improve image quality (sometimes with mixed success). The second phase dove into feature extraction, where we didn't just stop at raw pixel intensity but tried to engineer extra features and then there was the matter of picking the right kernel, which wasn't always straightforward. Finally, phase three was about solving the optimization problem and pulling everything together so the classifier was ready to make predictions.

3 Pre-processing

The training set $\tau = \{(x^{(i)}, y^{(i)})\}_{1 \leq i \leq m}$ contained 5000 grayscale images, each sized 28×28, evenly distributed across the 10 classes $\{0, 1, \ldots, 9\}$. To refine the data, I experimented with skew correction by applying the Hough transform to vertically align the dominant line. It was an interesting approach, as the alignment often depended on the clarity of the dominant line in each image.

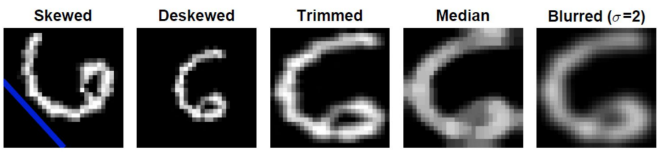

Figure 1. Effect of trimming the black border on the input images.

There is another pre-processing technique that I noticed helped improve the classifier's performance was trimming the image's black border, as illustrated in Fig. 1. Following this, I applied both a median filter, to denoise the image, and a Gaussian blur, to reduce intra-class variance, in parallel. The final pixel intensities were scaled to lie within the range $[0, 1]$. The interesting thing is that despite this pre-processing, global normalization of the features did not lead to any significant improvement in learning performance.

4 Features extraction

4.1 HOG features

I wanted to include some information about the image gradients, so I ended up trying Histograms of Oriented Gradients (HOG) features along with the raw pixel data. From what I could gather, these features are essentially a way to capture how edges are oriented, but in a somewhat nonlinear way. So, first, I worked out the vertical and horizontal gradients for the image. Then, I grouped those gradients into small, local patches to avoid getting too caught up in where the edges were exactly. After that, for every $b \times b$ block, I built an n-bin histogram that recorded the gradient orientations. Once I had all those histograms, I put them together into one long feature vector.

4.2 The kernels

For the raw and HOG features, I tested the following kernels:

Linear	$K(x, y) = x^T y$	Raw, HOG
RBF	$K_\sigma(x, y) = \exp\left(-\frac{x^T y}{2\sigma^2}\right)$	Raw, HOG
Poly	$K_d(x, y) = (1 + x^T y)^d$	Raw
χ^2	$K(x, y) = \sum_i \frac{(x_i - y_i)^2}{x_i + y_i}$	HOG
Intersection	$K(x, y) = \sum_i \min(x_i, y_i)$	HOG

Table 1. Tested Kernels for Raw and HOG Features

Implementing these kernels was fairly straightforward, but I noticed that both the intersection and χ^2 kernels turned out to be pretty heavy in terms of computation. This made them less efficient, even though they work quite well with HOG features.

5 Support vector machines

5.1 Sequential Minimal Optimization

I aim to solve the QP dual problem of the regularized SVM:

$$\begin{aligned} &\text{maximize } W(\alpha) = \mathbf{1}^T \alpha - \frac{1}{2} \alpha^T H \alpha \\ &\alpha \\ &\text{subject to } 0 \preceq \alpha \preceq C\mathbf{1}, \\ &\mathbf{Y}^T \alpha = 0 \end{aligned} \tag{QP1}$$

where $Y = [y^{(1)}, \ldots, y^{(m)}]$ and $H = \text{diag}(Y) K \text{diag}(Y)$. For a vector x, the prediction is made as:

$$y(z) = \text{sign}\, f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b.$$

A point is considered optimal if and only if the KKT conditions are satisfied:

$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)} f(x^{(i)}) \geq 1, \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)} f(x^{(i)}) = 1, \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)} f(x^{(i)}) \leq 1. \end{aligned} \tag{KKT}$$

I implemented a simplified version of the SMO algorithm to solve (QP1):

1. Repeat until convergence:

(a) Select a violating term α_i (i.e., one that does not satisfy (KKT)).

(b) Optimize $W(\alpha)$ with respect to α_i and a randomly selected term α_j, while keeping all other components fixed.

The algorithm is said to have converged when α doesn't change anymore, even after running several iterations (the full algorithm is in the appendix, if you're curious). I did try using some more complicated heuristics to choose the most violating pair (α_i, α_j) or even larger sets to work with, but honestly, this version of SMO seemed efficient enough for what I needed.

5.2 Multiple kernel learning (MKL)

Given a set of base kernels $\{K_k\}_{1 \le k \le n}$, the goal in linear Multiple Kernel Learning (MKL) is to figure out both the SVM parameters and the best linear combination of these base kernels at the same time:

$$K = \sum_{i=1}^{n} d_i K_i.$$

The resulting QP problem is:

$$\begin{aligned} \text{maximize} \quad & \mathbf{1}^T\alpha - \frac{1}{2}\sum_{i=1}^{n} d_i \alpha^T H_i \alpha + \frac{\lambda}{2}\|d\|^2 \\ \text{subject to} \quad & 0 \preceq \alpha \preceq C\mathbf{1}, \\ & \alpha^T Y = 0, \end{aligned} \tag{QP2}$$

Here, for every kernel, $H_i = \text{diag}(Y)K_i\text{diag}(Y)$. From my experience, SMO turned out to be a good fit for solving this problem.

6 Experiments and Results

When comparing the models, I decided to use k-fold cross-validation with $k = 5$. This way, every observation could take a turn in both the training and validation sets. The sections that follow go into more detail about the accuracy estimates from k-fold and how things played out on the Kaggle leaderboard.

6.1 One vs. One

I knew that the default SVM algorithm solves binary classification problems and since this task involves multi-class classification (10 classes), I considered all possible classifiers between pairs of classes, resulting in $\binom{10}{2} = 45$ classifiers.

6.2 One vs. Rest

In this case, for each class, I trained a classifier where all remaining classes were treated as a single class. In this setup, the positive class is a minority. To mitigate the imbalance, I introduced two separate regularization parameters per class ($C^+ = \beta C$, $\beta > 1$) to appropriately weight the classes.

6.3 Final Prediction

With either the one-vs.-one or one-vs.-rest strategy, multiple scores are generated for each observation. To combine these scores and produce a final prediction, I used the following methods:

- **One vs. One:** Each classifier casts a vote for its predicted class, and the final prediction is the class with the most votes.
- **One vs. Rest:** To compare the 10 scores, I estimated the posterior class probability for each class using Platt's scaling technique:

$$P(y = 1|x) \approx p_{a,b}(s) \equiv \sigma(as + b),$$

where s is the SVM score, and a, b are optimized using logistic regression.

6.4 Results

Kernel	Parameters	Acc(1)	Acc(2)
Raw (linear)	$C = 10, \beta = 1$	87.30%	
Raw (poly, $d = 3$)	$C = 10, \beta = 1$	95.80%	
Raw (poly, $d = 5$)	$C = 10, \beta = 1$	95.10%	
Raw (rbf, $\sigma = 4$)	$C = 10, \beta = 2$	95.70%	
Raw (rbf, $\sigma = 7$)	$C = 10, \beta = 2$	96.10%	95.42%
Raw (rbf, $\sigma = 10$)	$C = 10, \beta = 2$	95.90%	
HOG (linear)	$C = 10, \beta = 1$	92.30%	
HOG (intersection)	$C = 10, \beta = 1$	93.70%	
HOG (rbf, $\sigma = 2$)	$C = 10, \beta = 1$	96.10%	
HOG (rbf, $\sigma = 4$)	$C = 10, \beta = 1$	96.00%	
$K_1 =$ Raw (rbf, $\sigma = 7$) $K_2 =$ HOG (rbf, $\sigma = 4$) $K = K_1 \odot K_2$	$C = 10, \beta = 2$	96.90%	
$K_1 =$ Raw (rbf, $\sigma = 7$) $K_2 =$ HOG (rbf, $\sigma = 4$) $K_3 =$ HOG (linear) $K = K_1 \odot (K_2 + K_3) + K_2 \odot K_3$	$C = 10, \beta = 3$		
With blurring and deskewing		97.10%	97.26%
Without blurring nor deskewing		97.00%	97.08%
Without trimming		94.90%	
MKL $\{K_1, K_2, K_3\}$		97.20%	96.90%

Table 2. Acc(1): k-fold accuracy - Acc(2): Kaggle accuracy

6.4.1 Best Performance Statistics

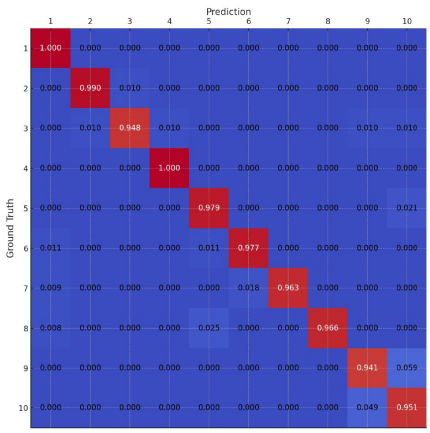

Figure 2. Performance comparison on selected metrics.

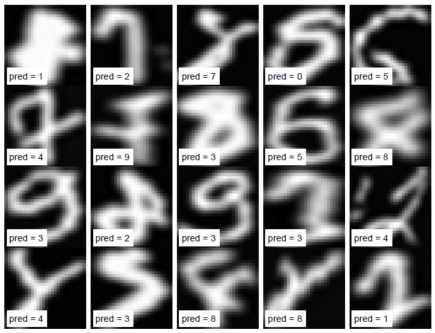

Figure 3. Some misclassified samples are unrecognizable even to a human classifier. Although applying a median filter might remove outlier elements like the character shown above, the overall performance drops.

7 Conclusion

On the MNIST problem, some classes, such as $\{0, 1, 3\}$, are more distinctive than others. Although the one-vs.-one strategy separates the classes accurately, I noticed that its overall accuracy is generally lower than the one-vs.-rest model with Platt's scaling. Among the tested kernels, the linear RBF, and exponential χ^2 kernels yielded better results on the HOG feature space, while the RBF kernel performed best on the raw pixel space.

My implementation of the MKL SMO algorithm was significantly slower than the standard SVM SMO. Additionally, I found that the elementwise product of kernels represents sample similarity better than the linear combination, which was the only version I tested with MKL.

References

[1] L. Bottou and C.-J. Lin, "Support vector machine solvers," in *Large Scale Kernel Machines*, pp. 301–320, 2007.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[3] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvements to platt's smo algorithm for svm classifier design," *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.

[4] H.-T. Lin, C.-J. Lin, and R. C. Weng, "A note on platt's probabilistic outputs for support vector machines," *Machine Learning*, vol. 68, no. 3, pp. 267–276, 2007.

[5] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods*, pp. 185–208, 1999.

[6] Z. Sun, N. Ampornpunt, M. Varma, and S. Vishwanathan, "Multiple kernel learning and the smo algorithm," in *Advances in Neural Information Processing Systems*, pp. 2361–2369, 2010.

Algorithm 1 SVM SMO

Inputs: Samples $(x^{(i)}, y^{(i)})_{1 \le i \le m}$, tolerance `tol`, regularization $C \in \mathbb{R}^m$
Outputs: Lagrange multipliers α, bias b
Initialization:
$\alpha = 0, b = 0$
$f = 0$ to store $f(x_i) = \sum_{j=1}^{m} \alpha_j y^{(j)} K(x^{(j)}, x^{(i)}) + b$
epoch $= 0$
while epoch < 10 **do**
    diff $\leftarrow 0$ ▷ Number of updated coefficients α_i
    for $i = 1, \ldots, m$ **do**
        Compute $E_i = f(x_i) + b - y^{(i)}$
        if $[y^{(i)} E_i < -\texttt{tol} \wedge \alpha_i < C^{(i)}] \vee [y^{(i)} E_i > \texttt{tol} \wedge \alpha_i > 0]$ **then**
            Select $j \in \{1, \ldots, m\} \backslash \{i\}$ and compute $E_j = f(x_j) + b - y^{(j)}$
            if $y^{(i)} = y^{(j)}$ **then**
                $L = \max(0, \alpha_i + \alpha_j - C^{(i)})$
                $H = \min(C^{(j)}, \alpha_i + \alpha_j)$
            else
                $L = \max(0, \alpha_j - \alpha_i)$
                $H = \min(C^{(j)}, C^{(i)} + \alpha_j - \alpha_i)$
            end if
            if $|L - H| < \texttt{tol}$ **then**
                continue to the next i
            end if
            $\eta = 2K(i, j) - K(i, i) - K(j, j)$
            if $\eta \ge 0$ **then**
                continue to the next i
            end if
            Update $\alpha_j = \alpha_j - y^{(j)} \frac{E_i - E_j}{\eta}$, crop to $[L, H]$
            if $|\alpha_j - \alpha_j| < \texttt{tol}$ **then**
                continue to the next i
            end if
            $\alpha_i = \alpha_i + y^{(i)} y^{(j)} (\alpha_j - \alpha_j)$
            Compute new bias b
            Update f, increment diff
        end if
    end for
    if diff = 0 **then**
        epoch++
    else
        epoch = 0
    end if
end while

Algorithm 2 MKL SMO

Inputs: Samples $(x^{(i)}, y^{(i)})_{1 \le i \le m}$, tolerance `tol`, regularization $C \in \mathbb{R}^m$, base kernels $\{K_l\}$
Outputs: Lagrange multipliers α, bias b, kernel coefficients d
Initialization:
$\alpha = 0, b = 0, f = 0, d = \mathbf{1}/n$
while epoch < 10 **do**
    diff $\leftarrow 0$ ▷ Number of updated coefficients α_i
    for $i = 1, \ldots, m$ **do**
        Compute $E_i = f(x_i) + b - y^{(i)}$
        if Conditions for update **then**
            Select $j \in \{1, \ldots, m\} \setminus \{i\}$
            for $1 \le l \le n$ **do**
                $A_l = K_l(i, i) + K_l(j, j) - 2K_l(i, j)$
                $B_l = y^{(i)} (\alpha Y)^T (K_l(i, :) - K_l(j, :))$
            end for
            Solve cubic equation to optimize Δ for α_i, α_j
            Update kernel coefficients d: $d = d + \frac{1}{2\lambda} \Delta B$
            Update f and compute b
            Increment diff
        end if
    end for
    if diff = 0 **then**
        epoch++
    else
        epoch = 0
    end if
end while