Predicting Hand-written Digits using Support Vector Machines (SVM)

Yue WU

yw9998

CS 391L

2019
March

1 Introduction

This homework aims at using Support Vector Machines (SVM) to detect hand-written digits automatically. We will use a certain amount (ntrain) of training images included in a training set of 60000 images to derive an algorithm that can detect the digits shown in a test set of 10000 images. Each image contains 784 pixels.

2 Methods

As Andrew Ng said in his lecture, "SVMs are among the best 'off-the-shelf' supervised learning algorithms." The algorithm not only tries to divide data into two groups using a linear classifier (or a hyperplane classifier for higher dimensions) as shown in Equation 1, it also aims at maximizing the margin between the two groups (Equation 2).

$$f(x) = \omega^T x + b \tag{1}$$

$$\frac{\omega}{\|\omega\|} \cdot (x_+ - x_-) = \frac{2}{\|\omega\|} \tag{2}$$

Therefore, we arrive at the following optimization problem:

$$\begin{aligned} \min_{\omega,b} \quad & \frac{1}{2}\left\|\omega\right\|^2 \\ y^{(i)}(\omega^T x^{(i)}+b) \geq 1,\ & i=1,...,m \\ g_i(\omega) = -y^{(i)}(\omega^T x^{(i)}+b)+1 \leq 0 & \end{aligned} \tag{3}$$

We also tried to reduce sensitivity to outliers by modifying the optimization problem as:

$$\begin{aligned} \min_{\gamma,\omega,b} \quad & \frac{1}{2}\left\|\omega\right\|^2 + C\sum_{i=1}^{m}\xi_i \\ y^{(i)}(\omega^T x^{(i)}+b) \geq 1-\xi_i,\ & i=1,...,m \\ \xi_i \geq 0,\ & i=1,...,m \end{aligned} \tag{4}$$

When C is smaller, we call it a softer margin. When C is larger (C = 10000), we have a harder margin. The dual optimization problem is:

$$\begin{aligned} \max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y^{(i)}y^{(j)}\alpha_i\alpha_j\left\langle x^{(i)}x^{(j)}\right\rangle & \\ 0 \leq \alpha_i \leq C,\ i=1,...,m & \\ sum_{i=1}^{m}\alpha_i y^{(i)} = 0 & \end{aligned} \tag{5}$$

For each image, values for the pixels are stored in a matrix named X (ntrain - 784), and the numbers of the group that each image belongs to are stored in a matrix named Y. The numbers stored in Y are called "labels." We will first train the algorithm by using the training set to calculate ω and b; then we can use Equation 6 to predict the labels for the test set, and compare our prediction Z with actual labels Y to calculate the accuracy of the algorithm. Note that there is a kernel in the equation. In our study, we tried two kinds of kernels: polynomial kernels (Equation 7) and Gaussian kernels (Equation 8), and compared the difference in accuracy.

$$z = \omega^T x + b \tag{6}$$

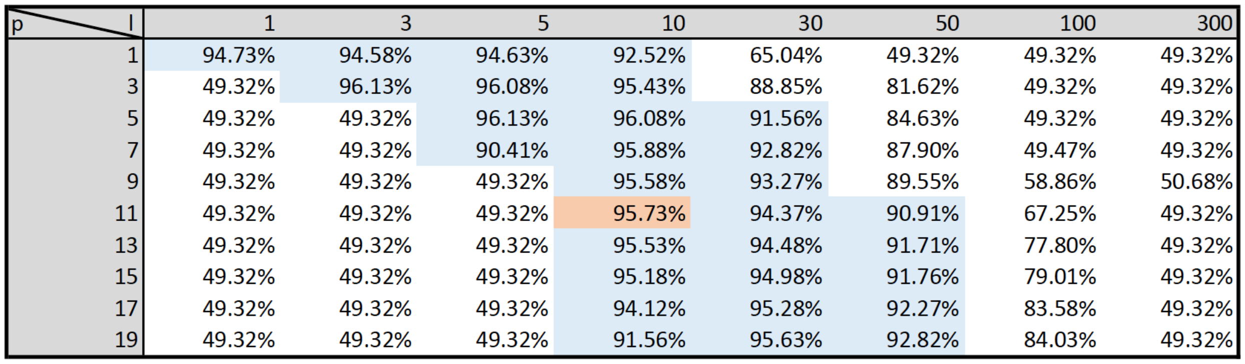
$$K(x,z) = (\frac{1}{l^2}x^T z + 1)^p \tag{7}$$

$$K(x,z) = exp\left(-\frac{\left\|x-z\right\|^2}{2\sigma^2}\right) \tag{8}$$

We also applied the Sequential Minimal Optimization (SMO) algorithm during the training. The SMO algorithm gives an efficient way of solving the dual problem of the SVM optimization problem (Equation 5). By applying the SMO algorithm, we 1) select two α parameters randomly, 2) find α_j so as to maximize the objective function, 3) solve for α_i using the value of α_j, 4) calculate the b threshold.

3 Results

We first used the data set (4 vs 9), which is supposed to be one of the harder ones to classify, to determine the optimal values for sizes of both kernels. The code used is stored in `hw3_compareplsig.m`. The number of training set is set to be 5000. Results are shown in Figure 1 and figure 2. For polynomial kernels (Equation 7), as p increases, the values rapidly increases, therefore we need larger l to scale down the value in order to get reasonable results. In other words, l should increase with p. The optimal combination we chose here is p = 11 and l = 10. For Gaussian kernels (Equation 8), accuracy increases with σ until σ reaches 10, then accuracy starts to decrease. Therefore, we chose σ = 10 as the optimal value.

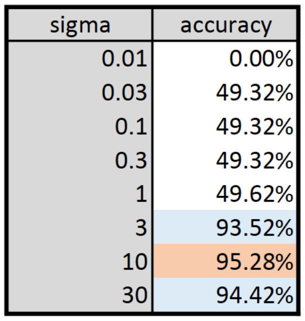


p \ l	1	3	5	10	30	50	100	300
1	94.73%	94.58%	94.63%	92.52%	65.04%	49.32%	49.32%	49.32%
3	49.32%	96.13%	96.08%	95.43%	88.85%	81.62%	49.32%	49.32%
5	49.32%	49.32%	96.13%	96.08%	91.56%	84.63%	49.32%	49.32%
7	49.32%	49.32%	90.41%	95.88%	92.82%	87.90%	49.47%	49.32%
9	49.32%	49.32%	49.32%	95.58%	93.27%	89.55%	58.86%	50.68%
11	49.32%	49.32%	49.32%	95.73%	94.37%	90.91%	67.25%	49.32%
13	49.32%	49.32%	49.32%	95.53%	94.48%	91.71%	77.80%	49.32%
15	49.32%	49.32%	49.32%	95.18%	94.98%	91.76%	79.01%	49.32%
17	49.32%	49.32%	49.32%	94.12%	95.28%	92.27%	83.58%	49.32%
19	49.32%	49.32%	49.32%	91.56%	95.63%	92.82%	84.03%	49.32%

(a)

Figure 1: Accuracies for different sizes of polynomial kernels. l increases from left to right, p increases from top to bottom. White color indicates low accuracy, blue color indicates high accuracy and orange indicate the combination we chose when applying the algorithm.

Next, we defined y = 1 and y = -1 for each case for both training set and test set



sigma	accuracy
0.01	0.00%
0.03	49.32%
0.1	49.32%
0.3	49.32%
1	49.62%
3	93.52%
10	95.28%
30	94.42%

(a)

Figure 2: Accuracies for different sizes of Gaussian kernels. σ increases from top to bottom. White color indicates low accuracy, blue color indicates high accuracy and orange indicate the value of σ we chose when applying the algorithm.

and chose Cp and Cr for each case. The code used is stored in `hw3_compareC.m`. The number of training set is set to be 2000. We manually input possible numbers for Cp and Cr (candidates) and the values for p, l, and σ that we just found out, trained the algorithm, predicted the labels and calculated error percentage for all combinations of Cp and Cr. The one with the highest accuracy is chosen to be the optimal combination for one data set. The candidates and optimal Cps and Crs are shown in Figure 3. We found out that for polynomial kernels, changes in C would not affect the accuracy much as the differences in error percentage between the softest margin and hardest margin are less than 1%. However, for Gaussian kernels, harder margin usually performs better. Differences in error percentage between the softest margin and hardest margin for easier cases (#1 and #2) are smaller, usually less than 6%. Differences in error percentage between the softest margin and hardest margin for harder cases (#3, #4 and #5) are larger, which can range from 50% for softest margin to 1% for hardest margin.

We then compared between optimal margins, hard margins, and soft margins. The code used is stored in `hw3_hardsoft.m`. We set the softer margin as $Cs = 0.01$, and the harder margin as $Ch = 1e5$. The number of the training set is set to be 5000. Values for p, l, and σ are the ones we calculated before. Figure 4 shows the differences in accuracy using different margins, different kernels and among five cases. We noticed that 1) for polynomial

Cp candidates	0.01	0.03	0.1	0.3	1	3	10	30	100
Cr candidates	0.01	0.03	0.1	0.3	1	3	10	30	100
p	11								
l	10								
sigma	10								

(a)

	case discription	Cp	Cr
#1	0 vs rest	3	3
#2	7 vs rest	30	10
#3	4 vs 9	0.3	10
#4	0 vs 8	0.03	1
#5	(0,8,3) vs (1,7,9)	0.3	3

(b)

Figure 3: (a) Candidates for Cp and Cr which the optimal values were chosen from, and the values of p, l, and σ we used during the tests. (b) The optimal Cps and Crs for all five cases chosen by the tests.

kernels, changes in C won't affect the accuracy much although the harder margins were slightly better, while for Gaussian kernels, harder margins performed better as we have seen before, 2) the easier cases (#1 and #2) generally shown higher accuracy than the harder cases (#3, #4 and #5), 3) accuracy differences between harder margins and softer margins were smaller for the easier cases (#1 and #2) compared to the harder cases (#3, #4 and #5).

	case discription	accu_poly_optimal	accu_poly_soft	accu_poly_hard	accu_rad_optimal	accu_rad_soft	accu_rad_hard
#1	0 vs rest	98.91%	98.33%	98.79%	99.37%	90.20%	99.46%
#2	7 vs rest	98.51%	97.87%	98.50%	98.23%	89.72%	98.16%
#3	4 vs 9	95.58%	94.93%	95.88%	95.28%	49.32%	95.43%
#4	0 vs 8	98.98%	98.11%	99.13%	98.57%	50.15%	98.72%
#5	(0,8,3) vs (1,7,9)	97.57%	97.05%	97.56%	96.69%	53.47%	96.43%

(a)

Figure 4: Accuracies for different sizes of Gaussian kernels. σ increases from top to bottom. White color indicates low accuracy, blue color indicates high accuracy and orange indicate the value of σ we chose when applying the algorithm.

4 Conclusion and discussion

The values for p, l, and σ can largely affect accuracy. For polynomial kernels, after finding a suitable p for a data set, the value of l should correlate well with p so that the values are best scaled. For Gaussian kernels, accuracy usually increases with σ to a maximum number, and then drops.

For our study, harder margins usually perform better. The reason might be that the data is easy to be distinguished from the others (the margins are already clear), in which case increasing tolerance for outliers will only cause more misclassification. Whether the margin is hard or soft affects Gaussian kernels much more than polynomial kernels, and differences in accuracy caused by different margins are larger for harder cases.

Easier cases generally show larger accuracy for harder cases, but this does not guarantee that our algorithm always performs better for easier cases because we also need to take account the total number of appearance of the target group in the whole data set. As we can see in Figure 4, when we used a softer margin for Gaussian kernel, the easier cases (#1 and #2) still shown accuracies as high as 90%, while accuracies for harder cases (#3, #4 and #5) were only 50%. It looks like the algorithm still succeeded in the easier cases, but it was not. For the easier cases, our target groups were 0 and 7, and both took up only 10% in the whole dataset. Even the algorithm did not predict any target number correctly we could still achieve an accuracy of 90%. Therefore, an accuracy of 90% for the easier cases and an accuracy of 50% for the harder cases both indicate failures in prediction.