Homework3: Support Vector Machine

Ke Wang
UT EID: kw28376

March 24, 2019

1 Introduction

The goal of this assignment is to use the support vector machine (SVM) algorithm to conduct digital recognition and test it on the MNIST dataset. SVM is a supervised learning algorithm which constructs a hyperplane in a high- or infinite-dimensional space that has the largest distance to the nearest training data of any class (also known as the functional margin). In this project, I implement the Support Vector Classifier (SVC) using the Sequential Minimal Optimization (SMO) algorithm described in Platt's paper [1]. I then compare the performance of my SVM algorithm with the PCA algorithm used in Homework 1.

2 Experiements

2.1 Formulation

Let us first consider a SVM classifier for a binary classification problem with labels y and features x. We use $y \in \{-1, 1\}$ to denote class labels. Parameterized with w and b, the classifier can be written as

$$h_{w,b} = g(w^T \phi(x) + b) \tag{1}$$

where $g(z) = 1$ if $z \geq 0$, and $\phi(x)$ denotes the feature vector. In the case that the dataset is not linearly separable, the optimization problem can be formulated as follows:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i \tag{2}$$

$$\text{s.t.} \quad y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \tag{3}$$

$$\xi_i \geq 0, i = 1, \ldots, m. \tag{4}$$

where the ith training exampleare allowed to have functional margin $1 - \xi_i$ (with $\xi_i >= 0$). Correspondingly, the cost function is increased by $C\xi_i$. The parameter C is a weighting constant.

We can then write the Lagrangian for the problem:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i[y_i(w^T\phi(x_i) + b) - 1 + \xi_i] - \sum_{i=1}^{m} r_i\xi_i \tag{5}$$

where the α_i's and r_i's are Lagrange multipliers (constrained to be non-negative). To implement the SMO algorithm, we need to solve the dual form of the problem:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle \tag{6}$$

$$\text{s.t.} \quad 0 \le \alpha_i \le C, \; i = 1, \ldots, m \tag{7}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \tag{8}$$

Here, $\langle x_i, x_j \rangle = K(x_i, x_j)$, and K is the kernel function. I test the performance of Gaussian kernel (Eq. 9) and polynomial kernel (Eq. 10) in this project.

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{9}$$

$$K(x, y) = (r + \sigma x^T y)^d \tag{10}$$

2.2 Multiclass SVM

In order to do multiclass classification, I use the 'one-vs-one' strategy. In other words, I train a Support Vector Classifier(SVC) for each pair of classes. Since we have ten different digits, the 'one-vs-one' ensemble will be composed of $C_{10}^2 = 45$ SVCs. The label will be determined from majority voting.

2.3 Comparison between different kernels

Five different kernels are used for the comparison of accuracy: (1) Gaussian kernel with $\sigma = 5$, (2) polynomial kernel with $r = 1, \sigma = 1, d = 1$, (3) polynomial kernel with $= 1, \sigma = 0.1, d = 2$, (4) polynomial kernel with $r = 1, \sigma = 0.01, d = 3$ and (5) polynomial kernel with $r = 0.3, \sigma = 0.01, d = 4$.

In the test, I fix the weighting constant C to 1. The number of training samples varies from 100 to 10000. The trained classifiers are used to classify 10000 test images. The accuracy obtained with different kernels and PCA algorithm is shown in Figure 1. The SVM algorithm with all kernels except the degree-1 polynomial kernel outperform the PCA algorithm. The highest accuracy (0.97) is achieved in this test with the degree-2 polynomial. However, we cannot conclude that this is the best kernel for this classification problem because I don't choose the best set of parameters for each kernel.

2.4 Hard and soft margin

Whether the margin is hard or soft can be controlled by the weighting constant C. When C is large, the SVM algorithm tends to classify all training examples correctly. The rigid margin will make the classifier sensitive to noise and outliers. Hard-margin SVM usually has a bad performance or fails when the data is not linearly separable in its feature space. When C is small, misclassified data points are allowed to make the classifier more generalizable.

Thus, in general, we can expect that soft-margin SVM has a better performance. I test the effect of soft/hard margin for SVM classifiers with the Gaussian kernel, 1-degree polynomial kernel, and 2-degree polynomial kernel as described in section 2.3. The accuracy results are listed in Table 1. Surprisingly, I do not observe a significant variation of classification accuracy with respect to the softness of the margin.

Kernel \ C	0.01	0.1	1	1000
Gaussian	0.7750	0.9360	0.9639	0.9698
1-degree polynomial	0.9285	0.9295	0.9173	0.9134
2-degree polynomial	0.9587	0.9684	0.9690	0.9684

Table 1: Accuracy achieved with different kernels described in section 2.3 and hard/soft margins.

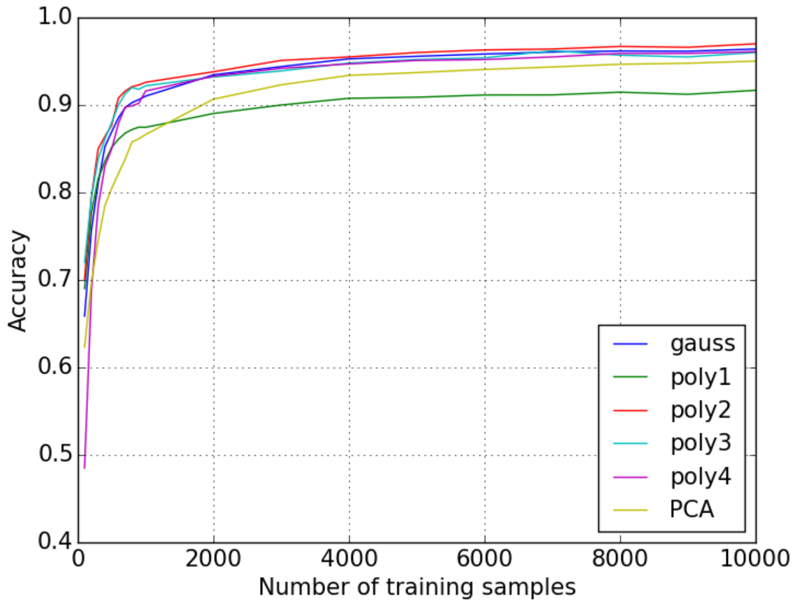

Figure 1: Comparison between the SVM algorithm with different kernels and the PCA algorithm. The same digit recoginition problem is solved using 6 classifiers: blue: SVM with gaussian kernel, green: SVM with 1-degree polynomial kernel, red: SVM with 2-degree polynomial kernel, cyan: SVM with 4-degree polynoial kernel, yellow: PCA.

2.5 Example classification queries

I test the required classification queries using the Gaussian kernel with $\sigma = 5$. All classifiers are trained using the first 10000 training images and tested on 10000 test images. The accuracy results are shown in Table 2

Query	Accuracy
0 vs rest	0.996
7 vs rest	0.989
4 vs 9	0.978
0 vs 8	0.992
(0,8,3) vs (1,4,9)	0.986

Table 2: Accuracy achieved with different kernels described in section 2.3 and hard/soft margins.

3 Discussion

In my implementation algorithm, I store the entire kernel matrix to accelerate the computation, which limits the number of examples that I can use for training. Moreover, the construction of the kernel matrix itself seems to be more time-consuming than the training stage. In my current python code, I use two for loops to compute the kernel matrix. I suspected that it was the reason why my code is slow. Thus I tried to use some matrix operations to replace the two for loops (see the commented code in `gaussianKernel.py`). However, the code became slower and required more storage. I also tried the standard SVM package from sklearn, which is 30 to 40 times faster than my implementation. In conclusion, there is a large room for improvement.

References

[1] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. page 21, April 1998.