

Support Vector Machine

Wang Zhiyuan 11610634

CSE

Computer Science and Technology

11610634@mail.sustc.edu.cn

1. Preliminaries

1.1. Software

In this project, I do some work to do the implement of the SVM [1] in two way, one is Gradient Descent [2] and another is SMO(Sequential Minimal Optimization) [3].

SVM is a very useful supervised learning model in the machine learning, it is used in many fields such as image recognition, object classification.

Traditional SVM have good effect in the case that the data is linear divisible, and if it need be used in the case that in the data is linear undivisible, it need use kernel function to process the data. In this report, the data I use will be linear divisible and the implement won't include the kernel function.

1.2. Algorithm

1.2.1. Overview. For the two algorithm I implement, the Gradient Descent is according to the loss function to do the Gradient Descent to reduce the loss. And the SMO is translating the problem $\min \frac{1}{2} \|\omega\|^2$ to its dual problem $\max \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j x_i^T x_j$

1.2.2. DataStructure. In the Gradient Descent, I just use numpy.ndarray to storage the the data to do the training data and the predict model. And in the SMO, I use numpy.matrix to storage the training data and the model, use the list to storage the model of the dual problem. And the set is used in the select the first parameter that need to be optimized to storage that the parameter can not be optimized

2. Methodology

2.1. Representation

2.1.1. Gradient Descent. In the Gradient Descent:

- Get the Gradient:

$$\begin{cases} -y \times x & 1 - y_i(\langle \omega, x_i \rangle + b) \leq 0 \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

- Do the moving

2.1.2. SMO. In the smo:

- Calculate the $m \times m$ matrix K, using the linear kernel
- Random get the first α , someone said that it should choice by the Heuristic function, but this way is too slow
- Get the second α by maximize the $|E_i - E_j|$
- Update the α_i and α_j , E and b

2.2. Architecture

2.2.1. Gradient Descent.

- training
 - *get_loss*
 - *move*
- predict

2.2.2. SMO.

- initialization
- training
 - *select_α_i*
 - *select_α_j*
 - *update_value*
- predict

2.3. Detail of Algorithm

2.3.1. Gradient Descent. First of all, we set two parameter: epochs and learningRate. The epochs means the times that we will repeat the processing of learning. The learningRate means the step size of each moving.

Then we will repeat epochs times, and random the sequence of the training data. Then move the hyperplane opposed the direction of the Gradient

The code of the algorithm is shown in the algorithm1.

2.3.2. SMO. [4] The first step, we get the list of all the α_i that $0 < \alpha < C$ and break the KKT, if there is no α have $0 < \alpha < C$ and break the KKT, then append all α that $\alpha == 0 || \alpha == C$ and break KKT, choice α_i in it randomly. If the list is empty, break the repeat.

algorithm 1 Gradiet Desent

```

1: function GD(epochs, learningRate, size, w)
2:   for i from 0 to epochs do
3:     index  $\leftarrow$  range(size)
4:     sort index randomly
5:     x  $\leftarrow$  tran[index]
6:     y  $\leftarrow$  label[index]
7:     loss  $\leftarrow$  0
8:     for xi, yi in zip(x, y) do
9:       loss  $\leftarrow$  loss + max(0, 1 - yi *  $\langle \mathbf{x}_i, \boldsymbol{\omega} \rangle$ )
10:      if y *  $\langle \mathbf{x}_i, \boldsymbol{\omega} \rangle$  < 1 then
11:         $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \text{learningRate} * y_i * \mathbf{x}_i$ 
12:      end if
13:    end for
14:    if loss == 0 then
15:      Break
16:    end if
17:  end for
18: end function

```

The second step, for the α_i we choice, find all the α left to find the α_j to $\max|E_i|$. Then we can calculate the range of the α_j . If the α_j out of the range, let $\alpha_j \leftarrow bound$, if $L \geq H$, we will return false. The next operation is update the α_j , in this operation, I need to get the η of the α_i and α_j , and then if the $\eta < 0$, this η is invalid and will return false. After update α_j successfully, we will update α_i , E and b.

We will do second step until all the α can not be optimized, or the process of update break down continuously for more than 500 times.

Most of the code is shown in the algorithm 2 to 6, and there are also some equation of the $K[i, j]$, E and b.

$$K[i, j] = \langle \mathbf{train}[i], \mathbf{train}[j] \rangle \quad (2)$$

$$E[i] = \langle \boldsymbol{\alpha} \times \mathbf{label}, K[i] \mathbf{label} \rangle + b - \mathbf{label}[i] \quad (3)$$

$$b_i^{new} = -E_i - y_i K[i, i] (\alpha_i^{new} - \alpha_i^{old}) - y_j K[j, i] (\alpha_j^{new} - \alpha_j^{old}) + b^{old} \quad (4)$$

$$b_j^{new} = -E_j - y_i K[i, j] (\alpha_i^{new} - \alpha_i^{old}) - y_j K[j, j] (\alpha_j^{new} - \alpha_j^{old}) + b^{old} \quad (5)$$

$$\begin{cases} b^{new} = b_i^{new} = b_j^{new} & 0 < \alpha_i^{new} < C, 0 < \alpha_j^{new} < C \\ b^{new} = \frac{b_i^{new} + b_j^{new}}{2} & otherwise \end{cases} \quad (6)$$

3. Empirical Verification

3.1. Design

In the verification of the SVM, I select the traing data on the sakai and divide it by 5 : 5, 6 : 4, 7 : 3, 8 : 2, 9 : 1 as the training data and the test data. Then training and predict them by Gradient Descent and SMO and record the result.

algorithm 2 Update α_j

```

1: function update $\alpha_j(i, j, H, L)$ 
2:    $\eta \leftarrow K[i, i] + k[j, j] - 2 * K[i, j]$ 
3:   if  $\eta \leq 0$  then
4:     return False
5:   end if
6:    $\alpha[j] \leftarrow \alpha[j] + \frac{\mathbf{label}[i] * (\mathbf{e}[i] - \mathbf{e}[j])}{\eta}$ 
7:    $\alpha[j] \leftarrow \text{Let } \alpha[j] \text{ not out Range}$ 
8:   updateE(j)
9: end function

```

algorithm 3 update α_i

```

1: function update $\alpha_i(i, j, \alpha_j \text{Old})$ 
2:    $\alpha[i] \leftarrow \alpha[i] + \mathbf{label}[i] * \mathbf{label}[j] * (\alpha_j - \alpha_j \text{Old})$ 
3:   updateE(i)
4: end function

```

3.2. Performance

The laptop I use is an Intel 6200U 2.8GHz@4 PC with 12GiB memory, for the test, it work just use battery, so the performance will be lower.

3.2.1. Hyper paramters. In the Gradient Descent, I set the *epochs* \leftarrow 1000, and the *learningRate* \leftarrow 0.1 In the SMO, I set the *C* \leftarrow 200, *iterTimes* \leftarrow 100, *toler* \leftarrow 0.00001 and *minMoving* \leftarrow 0.01

3.3. Result

TABLE 1. RESULT OF TRAIN DATA ON SAKAI WITH LINEAR KERNEL

Algorithm	Train num	Test num	Iter Times	Wrong Rate	Time use
GD	1200	134	1000	0.004	0.9
GD	1067	267	1000	0.002	0.4
GD	933	301	1000	0.005	0.4
GD	800	534	1000	0.0083	0.26
GD	667	667	1000	0.0101	0.2
SMO	1200	134	100	0.04	18
SMO	1067	267	100	0.04	15
SMO	933	301	100	0.04	11
SMO	800	534	100	0.055	9
SMO	667	667	100	0.05	7

The table 1 show the result that I test by the Gradient Descent and SMO.

3.4. Analysis

From the result in the table we can know, for a 10 demensions training data, the Gradient Descent can have great Correct Rate and high speed.

But by the analysis of the code, we can find, in the Optimization processing of the SMO, the number of demensions of the training data nearly have no influence for the training speed, the speed of the SMO depend on the number of the training data. for the Gradient Descent, we can find it calculate the $\boldsymbol{\omega}$ directly, so it will calculate slowly if there is a large number of demension training data.

algorithm 4 Change α

```
1: function CHANGEALPHA( $i$ )
2:    $E_i = e[i]$ 
3:   if ( $label[i] * E_i < -toler$  and  $\alpha[i] < C$ ) or
      ( $label[i] * E_i > toler$  and  $\alpha[i] > 0$ ) then
4:      $j, E_j \leftarrow select\alpha_j(i, E_i)$ 
5:      $\alpha_jOld \leftarrow \alpha[j].copy()$ 
6:     if  $label[i] == label[j]$  then
7:        $L \leftarrow max(0, \alpha[i] + \alpha[j] - C)$ 
8:        $H \leftarrow min(C, \alpha[i] + \alpha[j])$ 
9:     else
10:       $L \leftarrow max(0, \alpha[j] - \alpha[j])$ 
11:       $L \leftarrow max(C, C + \alpha[j] + \alpha[i])$ 
12:    end if
13:    if  $L == H$  then
14:      return 0
15:    end if
16:     $flag \leftarrow update\alpha_j(i, j, H, L)$ 
17:    if  $!flag$  then
18:      return 0
19:    end if
20:     $update\alpha_i(i, j, \alpha_jOld)$ 
21:     $updateB(i, j)$ 
22:    return 1
23:  else
24:    return 0
25:  end if
26: end function
```

algorithm 5 select α_j

```
1: function select $\alpha_j(i, E_i)$ 
2:    $max \leftarrow 0$ 
3:    $index \leftarrow -1$ 
4:   for  $j$  in  $range(size)$  do
5:      $temp \leftarrow abs(e[j] - E_i)$ 
6:     if  $temp > max$  then
7:        $max \leftarrow temp$ 
8:        $index \leftarrow j$ 
9:     end if
10:  end for
11:  if  $index == -1$  then
12:     $index \leftarrow random(size - 1)$ 
13:    while  $index == i$  do
14:       $index \leftarrow random(size - 1)$ 
15:    end while
16:  end if
17:  return  $index, e[index]$ 
18: end function
```

algorithm 6 SMO

```
1: function SMO( $maxIter, C, toler, minMove, size$ )
2:    $iters \leftarrow 0$ 
3:    $notUpdate \leftarrow 0$ 
4:    $over \leftarrow False$ 
5:   while  $iters < maxIter$  do
6:      $index \leftarrow set()$ 
7:     for  $i$  in  $range(size)$  do
8:       if  $0 < \alpha[i] < C$  then
9:          $temp \leftarrow abs(label[i] * g[i] - 1)$ 
10:        if  $temp > 0$  then
11:           $index.add(i)$ 
12:        end if
13:      end if
14:    end for
15:    if  $len(index) == 0$  then
16:      for  $i$  in  $range(size)$  do
17:        if  $\alpha[i] == 0$  then
18:           $temp \leftarrow 1 - label[i] * g[i]$ 
19:          if  $temp > 0$  then
20:             $index.add(i)$ 
21:          end if
22:        else if  $\alpha[i] == 0$  then
23:           $temp \leftarrow 1 - label[i] * g[i]$ 
24:          if  $temp > 0$  then
25:             $index.add(i)$ 
26:          end if
27:        end if
28:      end for
29:    end if
30:    if  $len(index) == 0$  then
31:      Break
32:    end if
33:    while  $len(index) \neq 0$  do
34:       $i \leftarrow index.pop()$ 
35:       $flag = self.changeAlpha(i)$ 
36:      if  $flag == 0$  then
37:         $notUpdate \leftarrow notUpdate + 1$ 
38:      else
39:         $notUpdate \leftarrow 0$ 
40:         $iters \leftarrow iters + 1$ 
41:      end if
42:      if  $notUpdate > 500$  then:
43:         $over \leftarrow True$ 
44:        Break
45:      end if
46:      if  $iters \geq maxIter$  then
47:        Break
48:      end if
49:    end while
50:    if  $over$  then
51:      Break
52:    end if
53:  end while
54: end function
```

References

- [1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [2] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.
- [3] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.
- [4] J. C. Platt, "12 fast training of support vector machines using sequential minimal optimization," *Advances in kernel methods*, pp. 185–208, 1999.