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 recognization, object classification.

Trainditional 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 accroding 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 $max\Sigma_{i=1}^{m}\alpha_i - \frac{1}{2}\Sigma_{i=1}^{m}\Sigma_{j=1}^{m}\alpha_i\alpha_j y_i y_j x_i^T x_j$

1.2.2. DataStructrue. 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:

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

- 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
  - $select\_\alpha_i$
  - $select\_\alpha_j$
  - $update\_value$
- predict

2.3. Detail of Algorithm

2.3.1. Gradiet 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.

Algorithm 1 Gradiet Desent

```
function GD(epochs, learningRate, size, w)
    for i from 0 to epochs do
        index ← range(size)
        sort index randomly
        x ← tran[index]
```

```
        y ← label[index]
        loss ← 0
        for xi, yi  in  zip(x, y) do
            loss ← loss + max(0, 1 − y_i ∗ ⟨x_i, ω⟩)
            if y ∗ ⟨x_i, ω⟩ < 1 then
                ω ← ω + learningRate ∗ y_i ∗ x_i
            end if
        end for
        if loss == 0 then
            Break
        else
            Change  learnRate  by  loss
        end if
    end for
end function
```

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.

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 $L >= H$, 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.

$$K[i,j] = \langle \boldsymbol{train[i]}, \boldsymbol{train[j]} \rangle \tag{2}$$

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

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

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

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

Algorithm 2 Update α_j

```
function updateα_j(i, j, H, L)
    η ← K[i,i] + k[j,j] − 2 ∗ K[i,j]
    if η ≤ 0 then
        return False
    end if
    α[j] ← α[j] + label[i]∗(e[i]−e[j]) / η
    α[j] ← Let  α[j]  not  out  Range
    updateE(j)
end function
```

Algorithm 3 update α_i

```
function updateα_i(i, j, α_jOld)
    α[i] ← α[i] + label[i] ∗ label[j] ∗ (α_j − alpha_jOld)
    updateE(i)
end function
```

Algorithm 4 Change α

```
function CHANGEALPAHA(i)
    E_i = e[i]
    if (label[i] ∗ E_i < −toler  andα[i] < C) or
(label[i] ∗ E_i > toler  and  α[i] > 0) then
        j, E_j ← selectα_j(i, E_i)
        α_jOld ← α[j].copy()
        if label[i] == label[j] then
            L ← max(0, α[i] + α[j] − C)
            H ← min(C, α[i] + α[j])
        else
            L ← max(0, α[j] − α[j])
            L ← max(C, C + α[j] + α[i])
        end if
        if L == H then
            return 0
        end if
        flag ← updateα_j(i, j, H, L)
        if !flag then
            return 0
        end if
        updateα_i(i, j, α_jOld)
        updateB(i, j)
        return 1
    else
        return 0
    end if
end function
```

Algorithm 5 select α_j

```
function selecα_j(i, Ei)
    max ← 0
    index ← −1
    for j  in  range(size) do
        temp ← abs(e[j] − Ei)
        if temp > max then
            max ← temp
            index ← j
        end if
    end for
    if index == −1 then
        index ← random(size − 1)
```

```
        while index == i do
            index ← random(size − 1)
        end while
    end if
    return index, e[index]
end function
```

Algorithm 6 SMO

```
function SMO(maxIter, C, toler, minMove, size)
    iters ← 0
    notUpdate ← 0
    over ← False
    while iters < maxIter do
        index ← set()
        for i in range(size) do
            if 0 < α[i] < C then
                temp ← abs(label[i] ∗ g[i] − 1)
                if temp > 0 then
                    index.add(i)
                end if
            end if
        end for
        if len(index) == 0 then
            for i in range(size) do
                if α[i] == 0 then
                    temp ← 1 − label[i] ∗ g[i])
                    if temp > 0 then
                        index.add(i)
                    end if
                else if α[i] == 0 then
                    temp ← 1 − label[i] ∗ g[i])
                    if temp > 0 then
                        index.add(i)
                    end if
                end if
            end for
        end if
        if len(index) == 0 then
            Break
        end if
        while len(index)¿0 do
            i ← index.pop()
            flag = self.changeAlpha(i)
            if flag == 0 then
                notUpdate ← notUpdate + 1
            else
                notUpdate ← 0
                iters ← iters + 1
            end if
            if notUpdate > 500 then:
                over ← True
                Break
            end if
            if iters >= maxIter then
                Break
            end if
        end while
        if over then
            Break
        end if
    end while
end function
```

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.

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 $learningRate \leftarrow [0.05, 0.01, 0.005, 0.001]$ 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, like when we de the recognization of image by Gradient Descent.

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.