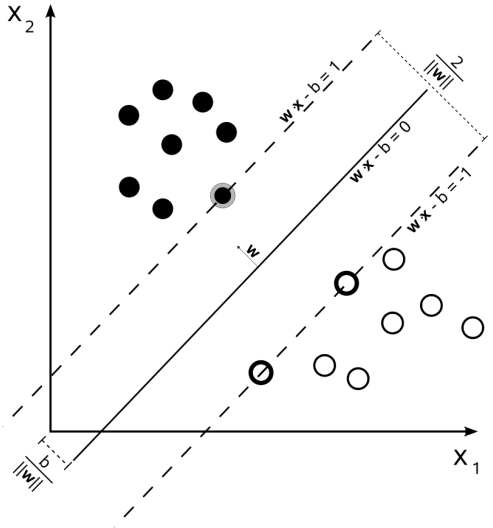
SVM Report

Yifei Xu 11611209
Southern University of Science and Technology

1. Preliminaries[1]

Classifying data is a common task in machine learning. Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a (p −1)-dimensional hyperplane. This is called a linear classifier. There are many hyperplanes that might classify the data. One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier it defines is known as a maximum margin classifier; or equivalently, the perceptron of optimal stability.



1.1 Introduction[1]

We are given a training dataset of ***n*** points of the form $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$ where the y_i are either 1 or −1, each indicating the class to which the point xi belongs. Each $\vec{x}_i$ is a p-dimensional real vector. We want to find the "maximum-margin hyperplane" that divides the group of points $\vec{x}_i$ for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point $\vec{x}_i$ from either group is maximized. Any hyperp $\vec{w} \cdot \vec{x} - b = 0$, en as the set of points x satisfying where ***w*** is the (not necessarily normalized) normal vector to the hyperplane. This is much like Hesse normal form, except that ***w*** is not necessarily a unit vector. The parameter $b / \|w\|$ determines the offset of the hyperplane from the origin along the normal vector ***w***.

And I implemented this linear SVM by SMO[2] algorithm and Gradient Descent Method[3].

1.2 Application

There are various applications of the influence maximization problem. In the real life, advertisement companies can find an optimal model to determine and classify the input.

2. Methodology

This part is to introduce the details of algorithm and the architecture of my program-Gradient Descent Method.

2.1 Notation

Notation	Description
n, m	n is the number of training data and m is the dimension.
learning_rate	the rate of learning, and learning is faster if rate is higher.

2.2 Data Structure

- *Worker*
  Pipeline processors
- *Network*
  Store the information of a network dataset.
  e.g. The weight of the edge *e* from *a* to *b*.
- *OptModel*
  Store the information of a model.
  e.g. x array, learning rate and so on.

2.3 Model Design

As following are the steps of solving a carp problem.

1. Read and resolve the training data.
2. For each test data Calculate the loss by using current w.
3. Get the loss and update w.
4. Repeat step 2,3 until we reach the termination condition.
5. Read and resolve the test data.
6. Output the prediction of test data.

2.4 Details of Algorithms

This part is to introduce the details of two training model(Gradient descend[3] and SMO[2]).

2.4.1 Gradient descend

Train : This method is to train the model and find the optimal ***w***, which includes ***b*** by adding an additional column to w.

Input: *x :* training data, *y :* labels
Output: *w :* optimal constant

---

counter ← 0
w ← a (*n*,1) matrix with random elements.
repeat
  counter ++
  resort *x* and *y* in random order
  for *xi, yi* in *x, y* **do**
    w ← ***cal_sgd****(xi, yi, w, counter)*
until *termination condition*
return *w*

***cal_sgd* :** This method is to update w based on the learning rate.

Input: x_i*:* matrix, y_i*:* matrix, *w:* matrix, *counter:* int
Output: *w :* matrix

---

if *y * x · w < 1* **then**
// using algorithms like simulated annealing to find the optimal w
  w ← *w - counter*$^{1/5}$ × *learning_rate*× (-*y* × *x*)
else
  w ← *w*
return *w*

2.4.2. SMO

Functions used in pseudo code are as following.

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \tag{1}$$

$$\begin{array}{ll} \text{If } y^{(i)} \neq y^{(j)}, & L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i) \\ \text{If } y^{(i)} = y^{(j)}, & L = \max(0, \alpha_i + \alpha_j - C), \quad H = \min(C, \alpha_i + \alpha_j) \end{array} \tag{2}$$

$$\eta = 2\langle x^{(i)}, x^{(j)} \rangle - \langle x^{(i)}, x^{(i)} \rangle - \langle x^{(j)}, x^{(j)} \rangle. \tag{3}$$

$$\alpha_j := \alpha_j - \frac{y^{(j)}(E_i - E_j)}{\eta} \tag{4}$$

$$\alpha_j := \begin{cases} H & \text{if } \alpha_j > H \\ \alpha_j & \text{if } L \leq \alpha_j \leq H \\ L & \text{if } \alpha_j < L. \end{cases} \tag{5}$$

$$\alpha_i := \alpha_i + y^{(i)} y^{(j)} (\alpha_j^{(\text{old})} - \alpha_j) \tag{6}$$

$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{(\text{old})})\langle x^{(i)}, x^{(i)} \rangle - y^{(j)}(\alpha_j - \alpha_j^{(\text{old})})\langle x^{(i)}, x^{(j)} \rangle. \tag{7}$$

$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{(\text{old})})\langle x^{(i)}, x^{(j)} \rangle - y^{(j)}(\alpha_j - \alpha_j^{(\text{old})})\langle x^{(j)}, x^{(j)} \rangle. \tag{8}$$

$$b := \begin{cases} b_1 & \text{if } 0 < \alpha_i < C \\ b_2 & \text{if } 0 < \alpha_j < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases} \tag{9}$$

SMO : This is to train the model and get optimal $\boldsymbol{\alpha}$ and $\boldsymbol{b}$.

Input:

C : regularization parameter
tol : tolerance
x : training data
y : labels

Output:

$\alpha \in R^m$ *:* Lagrange multipliers
$b \in R$: threshold

Initialize $\alpha_i \leftarrow 0, \forall i, b \leftarrow 0$.
repeat
  for $i \leftarrow 1$ to m
    $E_i \leftarrow \boldsymbol{f(}\, x^{(i)} \boldsymbol{)}$ - $y^{(i)}$ using function **(1)**
    if $(y_i E_i < -tol$ **and** $\alpha_i < C)$ **or**
      $(\, y_i E_i > tol$ **and** $\alpha_i > 0)$ **then**
      Select $i \neq j$ randomly
      $E_i \leftarrow \boldsymbol{f(}\, x^{(i)} \boldsymbol{)}$ - $y^{(i)}$ using function **(1)**
      $\alpha_{i(old)} \leftarrow \alpha_i$, $\alpha_{j(old)} \leftarrow \alpha_j$
      Compute L and H by using (2)
      if $L = H$ then
        continue for loop
      Compute η by (3)
      if $\eta >= 0$ **then**
        continue for loop
      $\alpha_i = \alpha_{inew}$ *by* using (4) and (5)
      if $|\, \alpha_j - \alpha_{j(old)} \,| < 10^{-5}$ **then**
        continue for loop
      Determine by α_i using (6)
      Compute b_1 and b_2 using (7) and (8)
      Compute b by (9)
  end for
until (termination condition)
return α and b

Calculate_w*:* This is to calculate w by using the result from ***SMO***.

Input: α : Lagrange multipliers, x: training data
y : labels

Output: $w \in R^n$

---

for $i \leftarrow 1$ to m
  $w \mathrel{+}= (\alpha_i \cdot y_i) * x_{[i:]}$
return w

---

3. Empirical Verification

This part is to introduce the performance of this algorithm on different datasets.

3.1 Dataset

- Willow recognization dataset (linear, 2000 for training and 500 for testing.)

3.2 Performance Measure

I. **Performance**

Fig.1 is the **Gradient descend's** performance on willow recognization datasets.

Fig.2 is the **SMO's** performance on willow recognization datasets.

Training Time (s)	Average Correction rate
20	98.14%
40	98.59%
60	99.44%
120	99.56%

Fig.1

Training Time (s)	Average Correction rate
20	97.90%
40	98.49%
60	99.24%
120	99.43%

Fig.2

II. **Test Environment**

CPU:	Intel Core i7
Frequency:	2 GHz
# Cpu:	1
# Core:	4
Python version:	3.7

3.3 Hyperparameters

Hyperparameter	Value
tol	0.01

According to practice, *tol = 0.01* is relatively suitable. But this based on the accuracy of the final result and the running time.

3.4 Experimental

According to Fig.1 and Fig.2, both Gradient Descend and SMO perform well. Because of the details of algorithms may be different from the pseudo code, the actual performance may be different.

3.5 Conclusion

Advantages : Thanks to TA's recommendation, the process of finding w and b is fast and the average correction rate is high.

Disadvantages : When it comes to a dataset with huge number of dimension, the correctness is unsatisfied.

4. References

[1] Wikipedia contributors. (2018, December 26). Support vector machine, from https://en.wikipedia.org/wiki/Support_vector_machine

[2] Platt, J.C., & Nitschke, R. (1998). Sequential minimal optimization : A fast Algorithm for Training Support Vector machines.

[3] Wikipedia contributors. (2018, December 26). Gradient descend, from https://en.wikipedia.org/wiki/Gradient_descent