Machine Learning Experiment with Support Vector Machine

Lin Wang, 11510064
School of Computer Science and Engineering,
Southern University of Science and Technology

January 4, 2019

1 Preliminaries

Support Vector Machine is a kind of significant method used for classification. In a general category, it belongs to supervised learning. SVM(Support Vector Machine) is first purposed by Vapnik and others people in 1992 and it has developed for nearly 30 years. Although its training speed is slow, due to its powerful modeling ability for complex nonlinear data, it is still widely used in many fields, including handwritten digital recognition, object recognition, benchmark time series prediction and so on. [1].

In the experiment, we mainly implement the algorithm of Gradient Descend to optimize the training processing. By using the providing data set, some character will be demonstrated.

1.1 Problem description

Support vector machines (SVMs) are a class of classification algorithms. To describe the problem in a simply way and more visual, the training set could be represented

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}, y_i \in \{-1, +1\}. \quad (1)$$

The basic idea of classification learning is to find a partition hyperplane in the sample space based on the training set and separate the samples of different categories. As can be see from the figure below, such hyperplanes are inifinite. Which one should I find?

In SVM problems, those could be classified as Linear separable SVM *(Data linearly separable, hard margin maximization, also known as hard-spaced SVM)*, Linear SVM *(When data is approximately linearly separable, soft margin maximization, also known as soft margin SVM)* and Nonlinear SVM *(When data is linearly inseparable, converted to soft-margin SVM by Kernels method.)*. In this project experiment, we only discuss linearly separable case.

Definition(Linear separable SVM): Given a linearly separable data set, it is equivalent to solve the corresponding convex quadratic programming problem by maximizing the margin. The separated hyperplane obtained by training is $\vec{w}^{\mathrm{T}} \cdot \vec{x} + b = 0$. And the corresponding classification decision function is $f(x) = sign(\vec{w}^{\mathrm{T}} \cdot \vec{x} + b)$. This is the linear separable SVM.

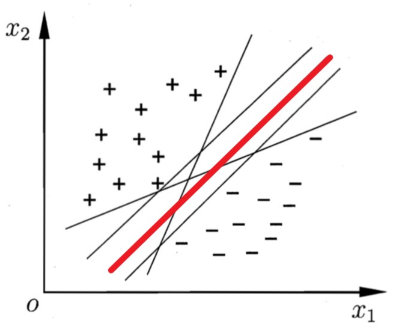


Figure 1: Two dimensional sample space, there are many kinds of hyperplane partition. Intuitively, we should look for: the division hyperplane located in the middle of the two types of samples. **That is, the red line above.**The hyperplane is best tolerant to local disturbances in the training samples. Look at the black line, the sample will be misclassified when it is slightly moved, in other words, the local disturbance to the sample is poorly tolerated. The red hyperplane produces the most robust results, with the strongest generalization ability for the unseen examples.

For this typical binary classification model, We all start with the hyperplane that divides the sample categories. According to the separating hyperplane function, $\vec{w}$ is a weight vector, $\vec{w} = (w_1, w_2, ..., w_m)$ is normal vector, which determine the direction of the hyperplane. b is the displacement term, which determines the distance between the hyperplane and the origin. In the sample space, the distance from any point x to the hyperplane is:

$$\hat{\gamma} = \frac{|\vec{w}^{\mathrm{T}} \cdot \vec{x} + b|}{\parallel \vec{w} \parallel} \quad (2)$$

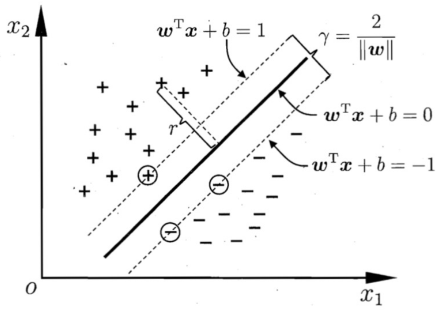
Suppose the hyperplane correctly classifies the training samples, so for $(\vec{x_i}, y_i) \in T$, if $y_i = +1$, then $\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b > 0$; if $y_i = -1$, then $\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b < 0$. The same as:

$$\begin{cases} \vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b \geq +1, & y_i = +1; \\ \vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b \leq -1, & y_i = 11; \end{cases} \quad (3)$$

As shown in the figure below, several training samples closest to the hyperplane make the equal sign of the above formula.

They are called "support vector". Each sample point corresponds to a support vector. The sum of the distances between two heterogeneous support vectors to the hyperplane is called margin (*simplified geometric margin*), the size is:

$$\gamma = \frac{2}{\parallel \vec{w} \parallel} \tag{4}$$



x_2
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=1$
$\gamma = \frac{2}{\|\boldsymbol{w}\|}$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=0$
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}+b=-1$
r
o
x_1

Figure 2: support vector and margin

Target: The basic idea of support vector machine learning is to solve the separated hyperplane that can correctly divide data set and have the largest margin. For a linearly separable training data set, the linear separable hyperplane has an infinite number of solutions, but the largest separated hyperplane is unique. The maximum margin is also call maximum hard-margin(*here, we will not discuss the soft-margin type*). When considering how to find the hyperplane with the largest margin, specifically, this problem can be expressed as the following constraint optimization problem:

$$\begin{aligned} \max_{\vec{w},b} \quad & \frac{2}{\parallel \vec{w} \parallel} \\ s.t. \quad & y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b) \geq 1, \quad i = 1, 2, \cdots, m \end{aligned} \tag{5}$$

Obviously, in order to maximize the interval, only need to maximize $\parallel \vec{w} \parallel^{-1}$, which equivalent to minimize $\parallel \vec{w} \parallel^2$, the original problem could be wroten :

$$\begin{aligned} \min_{\vec{w},b} \quad & \frac{1}{2} \parallel \vec{w} \parallel^2 \\ s.t. \quad & y_i(\vec{w}^{\mathrm{T}} \cdot \vec{x_i} + b) \geq 1, \quad i = 1, 2, \cdots, m \end{aligned} \tag{6}$$

This is the basic type of support vector machine.[2]

1.2 Software and configuration

This experiment is written in Python 3.7 and the only used library is `numpy`.

1.3 Algorithm

In this section, Gradient Descend algorithm was used. Because this is a kind of convex quadratic optimization problem and linearly separateable, there is a unique global feasible solution. The purpose of the training model is to find a suitable set of values for $\vec{w}$ and b so that the input values can reach the optimal separation after a series of transformations. In the gradient descned algorithm, we gradually update the weight matrix $\vec{w}$ by using the loss function that measures the difference between the predicted values and the target value. Initially, a random number generated could be regarded as the initial value of $\vec{w}$, b can be eliminated by panning $\vec{x}$.

Loss function: The larger the output value of the loss function is, the greater the difference is. Therefore, the training objective of SVM is to minimize the difference as much as possible.

$$L(\vec{w}, b) = \sum_{i=1}^{m} \max(0, 1 - y_i(< \vec{w}, \vec{x_i} > +b)) \tag{7}$$

For the value L calculated with the current $\vec{w}$ and b, if $L < 0$, which means the predicted value meet the requirement, loss is zero. If > 0, which indicates that the forecast is not accurate, the loss is the value.

Gradient descend: Reduce the loss by moving the loss value in the opposite direction of the current corresponding gradient. How much is moved at a time is determined by the learning rate.

$$\vec{w} = \vec{w} - \alpha \frac{\partial L(\vec{w}, b)}{\partial \vec{w}} \tag{8}$$

$$\frac{\partial L(\vec{w}, b)}{\partial \vec{w}} = \begin{cases} -y_i * \vec{x_i}, & if 1 - y_i(< \vec{w}, \vec{x_i} > +b) \geq 0, \\ 0, & otherwise. \end{cases} \tag{9}$$

2 Methodology

In this Experiment, we mainly use the test data set file (*train_data.txt*) to test. The data in the file are eleven numbers make a row. The first ten numbers are the size of the vector in the dimension as a sample. The last float number is +1 or -1, which is used to indicate the category of the sample point.

Because in the process of testing, we only have such training file contains all samples with their labeled category. We need to divide part of data set for testing.

2.1 Representation

Here are some interpretation for important variables which used in the program, more detail could be found in the **Detail of Algorithm** section:

- **data_mat:** the original data set represented as python numpy matrix. There are only vector demension data without labeled category.

- **label_mat:** the labeled part of our training data set. such like [1.0, 1.0, -1.0, ..., 1.0]. It also represent in numpy matrix with only one demesion.
- **test_data:** the test data set without labeled part. It is used for testing which have not appeared in the `data_mat`. Represented as numpy metrix.
- **epochs:** Number of iterations of gradient descent.
- **learning_rate:** The step size per move of Gradient Descend.
- **x:** the input value of gradient descend, represent as two demesion numpy array. The training data set.
- **y:** the labeled value of training data set represented as one demension numpy array.
- **w:** the representation of $\vec{w}$, initially, it is a random vector.
- **loss:** The loss value comes from loss function.

2.2 Architecture

Here is the project structure and function strategies. The main executed Python file is `SVM.py`, and the other attacted tools files was placed into the `util` folder. The solver create `class` object from other file and obtain final feasible solutions.

- **SVM.py**
  - `argv_parse`: A simply function to parse the system argument. In this experiment, the train_data and test_data are the location arguments, the "-t" optional argument will specify the termination condition of program.
  - `load_train_data`: load the given train data set and return the original vector sample part and labeled part respectively.
  - `load_test_data`: load the given test data set and return only the original vector part.
  - `get_gd_result`: a simple function to invoke the gradient descend class to finish training task and return the predicted value of test data set represented as numpy one demension array.
- **gd.py**
  - `get_loss`: Input the training data set and it corresponding labeled value, it is used to verify the accuracy of value of $\vec{w}$. It will return the loss value directly.
  - `cal_sgd`: Calculate the updated $\vec{w}$ value. For each iteration, if the sample is inside the two hyperplane, the value of $\vec{w}$ will be updated according to the learning rate.
  - `train`: The mainly training funtion. In the experiment, I adjusted the the judgment condition of loop iteration. Only when the loss value less or equal to zero, then the iteration stop, which ensure the finding hyperplane is able to meet all constraint of the train data sample.
- **file_apart.py**

  It is an extra python script which could be used to split original train data into training part and test part. Meanwhile, we can also modify the proportion of the test set in the original data which will influence the accuracy of rediction.

2.3 Detail of Algorithm

Here describe the detail of algorithm in some vital functions.

Algorithm 1 : Core training algorithm of gradient descend. Here are the mainly idea of algorithm, from the initialization to final updated $\vec{w}$. The algorithm described by state.

Variable $\vec{w}$ is a global variable. In the `While` loop, at each iteration, we calculate the data samples in all training sets to get a new $\vec{w}$ value.

Algorithm 1 train

```
Input: $x, y, w$
Output: $w$                                        ▷ Updated $\vec{w}$
 1: add one demesion data to $\vec{x}$                ▷ To eliminate $b$
 2: get random value for vector $\vec{w}$
 3: initial $loss \leftarrow 1$              ▷ To make it larger than 0 initially
 4: initial $epoch \leftarrow 0$
 5: while $loss > 0$ do
 6:     For all data samples, scramble the order
 7:     $loss \leftarrow 0$
 8:     for $xi \in \vec{x}, yi \in \vec{y}$ do
 9:         $loss \leftarrow loss +$ The loss from the current sample
10:         $\vec{w} \leftarrow$The value of $\vec{w}$ updated by the current sample
11:     end for
12:     $epoch \leftarrow epoch + 1$
13: end while
```

3 Empirical Verification

Empirical verification is mainly conducted by spliting the original training set file into different parts. Different proportion of them decides the size of training data which will influence the accuracy of predicted result.

3.1 Design

By research the data size of training set, learning rate or might be the random way. We likely get different results. The empirical varification is mainly analyse this difference to get conclusion.

3.2 Data and data structure

Data being used in this experiment is the *train_data.txt* file. The data structures used in this experiment imainly is numpy array, several python list are also used.

3.3 Performance

The performance mainly was verified by time use of the program. By using the `time` module, we calculate the time used from program start to end. Meanwhile, the number of iterations will also influence the performance.

3.4 Result

Here is a table which presents some performance indicator for experimence.

Proportion means the percentage of data used for training, and the **Performance** is a series of indicator to express the efficiency of such case. **Time used** is the time cost when the loss value is reduced to zero and the entire program is finished. **Number of iterations** refers to the number of iterations required to reduce the loss value to zero. **Accuracy** refers to the rate of correct of predicted values.

Performance \ Proportion	50%	60%	70%	80%	90%
Time used	1.70 s	4.92 s	5.09 s	13.18 s	11.54 s
Number of iterations	801	2101	1815	4003	3367
Accuracy	98.95%	99.81%	99.75%	99.62%	99.25%

Table 1: All test result for supply

The plotted chart as show here is the drop in loss values over multiple iterations, it is comes from the 80% proportion case.

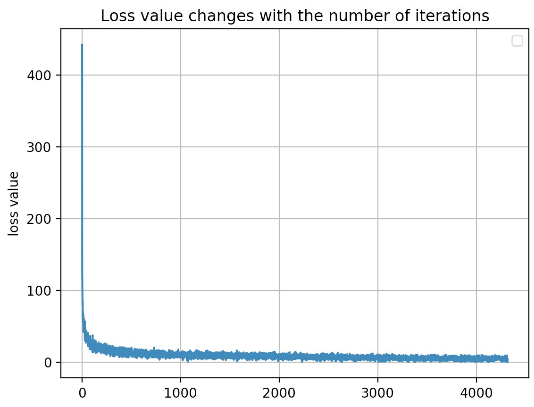


Figure 3: Loss value changes with the number of iterations

3.5 Analysis

In conclusion, as we can see it very intuitively from the chart, the loss value changes very quickly at the beginning and then in a very small range of variation in subsequent iterations. The volatility is samll. The loss value has reached a level close to zero after almost 200 to 300 iterations. So we can reduce the number of iterations to increase the time.

References

[1] Peter Harrington, *Machine Learning in Action*. POST & TELECOM Press 1th ed. 2013

[2] Li Hang, *Statistical learning method*. Tsinghua University Press 1th ed. 2012