SVM Report

Yilin Zhang 11510280

School of Computer Science and Engineering
Southern University of Science and Technology
11510280@mail.sustc.edu.cn

December 31, 2018

1. Preliminaries

Support vector machine (SVM) is supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. In reality, SVM is widly used in spam identification, market forecasting *etc*.

An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

In this project, we need to implement an SVM classifier named SVM.py. It's used to classify a set of linear data with ten dimensions by training on a labeled dataset.

1.1. Software

This project is written in Python. The libraries being used includes *argparse*, *Numpy* and *Random*. The local test environment is as follows:

- Python version: 3.6.6
- Operation System: macOS
- Processor: 3.5 GHz Intel Core i7
- Memory: 16 GB

2. Methodology

Given some data points, which belong to two different classes, a linear classifier is now to be found to divide the data into two categories.[1] X denotes data points, Y denotes the category (Y can take 1 or -1, representing two different classes, respectively), and the learning goal of a linear classifier is to find a hyperplane in the n-dimensional data space, a hyper-planar equation that can be expressed as

$$w^T x + b = 0 \tag{1}$$

This hyperplane can be represent with classification function,

$$f(x) = w^T x + b \tag{2}$$

when $f(x)$ is equal to 0, x is the point on the hyperplane, the point $f(x)$ greater than 0 corresponds to the data point $y = 1$, and the point f(x) less than 0 corresponds to the point $y = -1$.

2.1. Representation

SVM.py need to parse arguments from commend line and read *.txt files including the information of points or labels. I used a m-n matrix stores these points, where m denotes number of points and n denotes that each point has n features.

The format of SVM call are as follows (the test environment is Unix-like):

python SVM.py <train data> <test data> -t <time budget>

Output: predicted labels of points.

- **<train data>:** absolute path of the train data.
- **<test data>:** absolute path of the test data.

- **<time budget>:** a positive number which indicates how many seconds the algorithm can spend on this instance.

Some constants are as follows:

- **learning_rate:** Control the rate of gradient descent
- **precision:** Determine the maximum number of loss when the result is acceptable.
- **max_random_sample:** Determine the maximum number of random samples when stochastic gradient descent.

2.2. Architecture

The executable files is *SVM.py*, which contains parameter parsing, flie reading, and result printing. There is a modeule *gd.py* in the *utils* folder, offer a class GD and its functions get_loss, cal_sgd, train and predict.

- **SVM:**
  - **parse:** parse command line parameters ans return them.
  - **load_data_set:** read train file and test file, store data in arrays, respectively.
  - **output:** print result in format.
- **gd:**
  - **__init__:** init an object contains x, y, learning rate, precision, max random sample, and a random generated array w.
  - **get_loss:** calculate the loss value of x with label y in current prediction, use to compare the difference between the predicted value and the target value.
  - **cal_sgd:** update the weight matrix w with gradient descent.
  - **train:** train the model, which means update w until loss is small enough.
  - **predict:** predict test data and return their labels.

2.3. Algorithm

This linear SVM simply use gradient descent method to train the model. To avoid training too large datasets, using *max_random_sample* as a maximum for random training.

Here are details of these algorithms.

Algorithm 1 get loss

```
Input: $x_i, y_i$
Output: $loss$
1: $loss = max(0, 1 - y_i w {x_i}^T)$
2: return $loss$
```

Algorithm 2 stochastic gradient descend

```
Input: $x_i, y_i, w$
Output: $w$
1: if $y_i w {x_i}^T < 1$ then
2:     $w = w - learning\_rate * (-y_i x_i)$
3: end if
4: return $w$
```

3. Empirical Verification

3.1. Experiments

A basic test has been given to test the usability of SVM code, and it is running well. The loss value of each epoch as figure 1. We can see that the loss quickly descend at first and fluctuate at the state of stability. Actually, the model has a good performance with little loss, which spend a lot of time to avoid at last. Therefore, sometimes we can tolerate some loss to reduce the taking time.

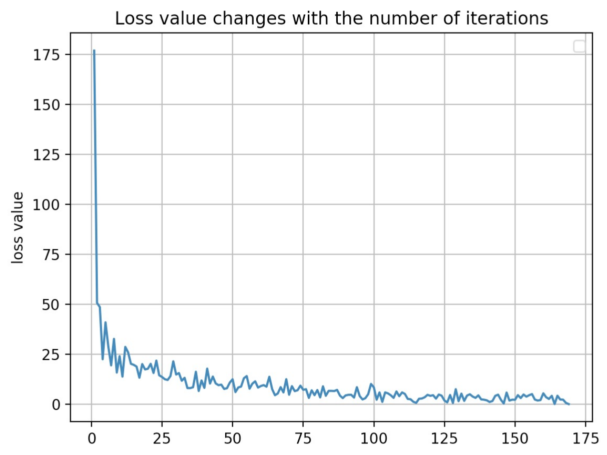

Figure 1: *loss value of each epoch*

3.2. Data and data structure

Data being used in this project is a labeled dataset to train and an unlabeled dataset to test.

Algorithm 3 train

```
Input: $timelimit$
 1: while $loss > precision$ do
 2:     if time limited then return
 3:     end if
 4:     while m > max random sample do
 5:         $m = \lfloor m/2 \rfloor$
 6:     end while
 7:     for random $mx_i$ do
 8:         $loss = loss + get\_loss(x_i, y_i)$
 9:         $w = cal\_sgd(x_i, y_i, w)$
10:     end for
11: end while
12: return $w$
```

Data structure used in this project includes array, tuple, and list.

3.3. Performance

This algorithm used stochastic gradient descend implements SVM, so its theoretic running time, computed by time complexity of functions is about $O(mn)$, where m is max random samples, n is the number of epochs. So even if the train dataset is large, we can use m to control training time, but with that comes a drop in accuracy.

3.4. Analysis

Gradient descent is a commonly used training method in machine learning. It is easy to understand and implement, but due to depends on the initial solution, it is easy falling into local optimum. The Sequential Minimal Optimization (SMO) algorithm [2], first proposed by John C. Platt, has a better performance for linear SVM and data sparsity. I implemented this method, but it performs not as good as the gradient in terms of time and prediction on the test dataset. On the one hand, it may be due to insufficient optimization of the code, and on the other hand, it may due to the parameter setting is not good enough. In addition, Python also implements this algorithm in the Sklearn package for common using.

Sometimes data is not linearly partitionable, and one way to handle it intuitively is to map the data to a high-dimensional space and then linearly divide it. In order to avoid tedious complex operations in high-dimensional space, we can introduce the technique of kernel function, which do calculation in advance on low-dimensional, and then express the substantive classification effect on high-dimensional.

The current algorithm does not implement these parts, it is the direction that can be further studied.

References

[1] July‘s blog

[2] John C. Platt. *"Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines"*, 1998.