Project 4: Support Vector Machine

Yuejian Mo 11510511
Department of Biology
Southern University of Science and Technology
Email: 11510511@mail.sustc.edu.cn

1. Preliminaries

Support vector machine, are 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. [1] If the dataset are points in space, we can find out an gap separate dataset into two part by using SVM.

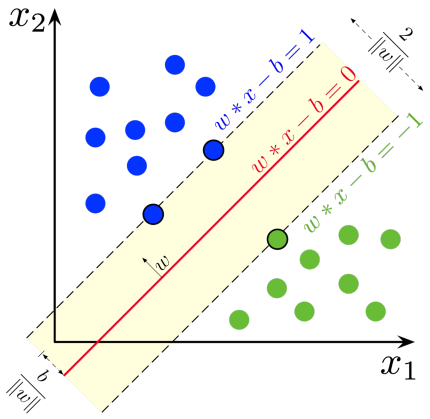

Figure 1. Separate data points into two part using SVM(from Larhmam)

The original SVM algorithm was invented by Vladir N. Vapnik and Alexey Ya. Chervonenkis in 1963. During the development, SVM are able to deal with more tasks. SVM are used in text and hypertext categorization, classification of images, hand-written characters recognition and proteins classification.

In this project, I use SVM with random gradient descent to classify training data almost correctly.

1.1. Software

This project is written by Python 3.7 with editor Vim. Numpy, os, time, random, sys and argparse library are used.

1.2. Algorithm

Model question with SVM. To optimal time cost and current of solution, random gradient descent is used.

2. Methodology

Training data is ....
SVM is modeled as ...
random ....
Training set contain 1334 sample, and each sample contain 10 features and 1 label. With the help of other mathematical tools, training data show that it can be classified in linear model. So, SVM model is used as training model.

For most common and basic SVM(Figure 1), linear SVM uses two parallel vector $w \cdot x - b = 1$ and $w \cdot x - b = -1$ to separate dataset x into two parts, where x is $m * n$ matrix with n training samples of m features, w is a m dimension vector of SVM parameters and b also is parameter of SVM. The rule is that gap between two support vector don't exist any points expect of on the vectors. To optimal correctness of classification, enlarging the distance $\frac{2}{||w||}$ between two support vector is one of method. It belong to convex optimization problem as following:

$$min_{w,b}\frac{||w||^2}{2}$$
$$s.t. y_i((w \cdot x_i) + b) \leq 1, i = 1, ..., n$$

To train data, we use loss function(also called cost function) to measure the difference between SVM's prediction and actual labels. Each training, SVM try to decrease the value of loss function.

$$L(w, b) = \sum_{i}^{n} = 1max(0, 1 - y_i((w_i, x_i) + b))$$

Then Gradient descend is used to update the feature weight matrix w. Each training, gradient descend promote loss function decrease by moving to opposite direction of gradient.

$$w = w - \alpha\frac{dL(w, b)}{dw}$$
$$\frac{dL(w, b)}{dw} =$$

Finally, the seed required for IMP is generated by a natural greedy algorithm. Each time, a node with highest influence will added to a seeds set. This empty seed set will fill with k node, where k is given from outside.

2.1. Representation

Some main data are maintain during process: **time_budget**, **node_num**, **graph_cp**, **graph_pc**, **incoming**. Others data would be specified inside functions.

- **x**: Numpy matrix to store input data.
- **y**: 1-d numpy matrix of label of input data.
- **w**: 1-d numpy matrix of weight and bias of SVM parameters.

2.2. Architecture

Here list main functions of **SVM.py** in given code:

- **init**: load train data to x, y; load time limit; initialize parameter matrix w.
- **cal_sgd**: Update w using gradient descent.
- **train**: Train w from train data data x, y.
- **predict**: Predict label of test data
- **__main__**: Main function

The **SVM.py** is executed locally.

2.3. Detail of Algorithm

Here describes some vital functions.

- **init**: load train data and initial parameters

Algorithm 1 int

Input: $input\_file\_name, time\_limit$
Output: $x, y, w, cycle$
open $input\_file\_name$ as $file$
set $data$
for each line in $file$ **do**
    split line, then package and append line info to $data$
end for
transform set $data$ to matrix x
$y \leftarrow x[:, -1]$
$x[:, -1] \leftarrow 1$
initialize w with length of $x[:, -1]$ and value of 0
$cycle$
return $x, y, w, cycle$

- **cal_sgd**:Update w using gradient descent

Algorithm 2 cal_sgd

Input: $xi, yi, w, ratio, learn\_speed$
Output: w_next
if $yi * (\cdot xiw) < 1$ **then**
    $w\_next = w - ratio * (learn\_speed * w - yi * xi)$
else
    $w\_next = w - ratio * learn\_speed * yi$
end if
return w_next

- **train**: Train w with data x, y

Algorithm 3 train

Input: $x, y, w, cnt, learn\_speed$
Output: w
$t \leftarrow 0$
while running time don't exceed ordered time **do**
    reorder x and x in the same time
    $loss \leftarrow 0$
    $t \leftarrow t + 1$
    $1//(t * learn\_speed)$
    for each line x_i and y_i in training data **do**
        $w = cal\_sgd(x_i, y_i, w, ratio, learn\_speed$
    end for
end while
return w

- **predict**: Predict the label of test data

Algorithm 4 predict

Input: $test\_file, w$
Output: $label\_set$
open $test\_file$ as $file$
create set $test\_data$
for each line in $file$ **do**
    split line, then package and append line info to $test\_data$
end for
transform set $test\_data$ to matrix $test\_set$
$test\_set[:, -1] \leftarrow 1$
create $label\_set$
$label\_set \leftarrow sgn(\cdot test\_setw)$
return $label\_set$

3. Empirical Verification

Empirical verification is verified offline with given data set. We do the OJ system. $ISE.py$ almost pass all dataset with reasonable bias. $IMP.py$ only was test with network-5-IC and provide reasonable seeds set.

3.1. Design

SVM classify data at less 90 percent robust.

Random gradient return in reasonable time than simple gradient down algorithm.

Independent cascade model return reasonable influence is small graph and big graph. But Linear Threshold model return less reasonable due with large graph. Finally, I found my code would added some element repeatedly result in larger bias.

For hill greedy, because I just evaluate once, it produce seeds set fast. In small graph, which produce seeds in 65 percent effect. But Not optimal seeds are produced every time.

3.2. Data and data structure

For the convenience, matrix are used widely to store train set, test set and features matrix.

3.3. Performance

Following table show different performance with different parameters. Offline test perform at Fedora 29 with $Intel^{®}$ Xeon(R) CPU E5-1680 v3@3.20GHz and 32GiB memory.

Dataset	Run Time(s)	Result
network-seeds-IC	23.23	5.015
network-seeds-LT	23.23	5.015
network-seeds2-IC	23.24	30.47
network-seeds2-LT	57.79	37.03
NetHEPT-5seeds-LT	58.55	341.9
NetHEPT-5seeds-IC	27.87	276.3
NetHEPT-50seeds-LT		
NetHEPT-50seeds-IC	27.87	1003
network-5-IC	0.73	19.45

3.4. Result

Best model classify data set at 99.6 percent of correctness.

3.5. Analysis

Because natural Hill Greedy is kind of greedy algorithm without heuristics, most of solutions are not optimal. During using adjacent list rather than adjacent matrix, space cost is more lower than $O(n^2)$. Because every point only is evaluate once, only less time it cost. Total, no more than $O(n^2)$ is required.

Acknowledgment

Thanks TA Yao Zhao who explain question and provide general method to solve it. And I also thanks for Kebing Sun discuss and share the idea about algorithm.

References

[1] Wikipedia