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(SVM) is... Our data ...

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.

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 down is used.

2. Methodology

Training data is ....

SVM is modeled as ...

random ....

Firstly, a social network is modeled as a directed graph $G = (V, E)$ and each edge $(u, v) \in E$ is associated with a weight $w(u, w) = \frac{1}{d_{in}}$ which indicates the probability that u influences v. $S \in V$ is the subset of nodes selected to initiate the influence diffusion, which is called seeds set.

Then, I try to evaluate the spread influence with two different diffusion models. In *Linear Threshold* model, a node v is influenced by each neighbor u according to a weight $w_{v,u}$ such that $\sum_u w_{v,u} <= 1$. The dynamics is following. Each node v are accessed a threshold $\theta \in [0, 1]$ randomly obeyed uniform distribution at first step; this represents the weighted fraction of v' neighbors that must become active in order for v become active. In step t, each inactive node v will be active by its active neighbors w at step $t - 1$ as following

$$\sum_w w_{u,v} >= \theta_v$$

. Once no new active node is generated, maximal spread influence will obtain with given seeds set in specified graph. I notices final influence as the total number of active node σ.

In *Independent Cascade* model, I start with an initial set of active nodes S, and the process unfolds in discrete steps according to the following randomized rule. When node v first becomes active in step t, it is given a single chance to activate each currently inactive neighbor w; it succeeds with a probability $w_{v,w}$, independently of the history thus far. Every step. In practical, for each inactive neighbor w of new active node v will be active, if $w_{v,w}$ is larger than a new generated random number between 0 and 1 obeyed uniform distribution. (If w has multiple newly activated neighbors, their attempts are sequenced in an arbitrary order.) If v succeeds, then it cannot make any further attempts to activate w in subsequent rounds. Again, the process runs until no more activation are possible.

Both of Linear Threshold and independent Cascade model are stimulated by repeating many time to average influence for each seed.

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: $x, y, w, cycle$
1: open $input\_file\_name$ as $file$ {open file and read line by line}
2: {Read each line information until arrive edge information}
3: set $data$
4: for each line in $file$ do
5:     split line, then package and append line info to $data$
6: end for
7: transform set $data$ to matrix $x$
8: $y \leftarrow x[:, -1]$
9: $x[:, -1] \leftarrow 1$
10: initialize $w$ with length of $x[:, -1]$ and value of 0
11: $cycle$
12: 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
2:      $w\_next = w - ratio * (learn\_speed * w - yi * xi)$
    else
4:      $w\_next = w - ratio * learn\_speed * yi$
    end if
6:  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
3:      reorder $x$ and $x$ in the same time
        $loss \leftarrow 0$
        $t \leftarrow t + 1$
6:      $1//(t * learn\_speed)$
        for each line $x_i$ and $y_i$ in training data do
            $w = cal\_sgd(x_i, y_i, w, ratio, learn\_speed$
9:      end for
    end while
    return $w$
```

- **hill_greedy**: Generate optimal seed using Hill Greedy Algorithm

Algorithm 4 hill_greedy

```
Input: $size, model$
Output: $seeds$
    create empty $ans\_seeds$
    $cnt \leftarrow 0$
    while $cnt \neq size$ do
4:      copy $ans\_seeds$ to $cur\_seed$
        $high \leftarrow 0$
        $point \leftarrow 1$
        for each node 4 don't be chosen do
8:          add $v$ to $cur\_seed$
            if $model$ == IC then
                $new\_high \leftarrow IC(graph\_pc, cur\_speed, incoming)$
            else
12:             $new\_high \leftarrow LT(graph\_cp, cur\_speed, incoming)$
            end if
            if $nw\_high \leq high$ then
                $high \leftarrow new\_high$
16:             $point \leftarrow v$
            end if
        end for
        add $point$ to $ans\_seeds$
20:     $cnt \leftarrow cnt + 1$
    end while
    return $ans\_seeds$
```

3. Empirical Verification

Empirical verification is confirmed in 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

Dictionaries and lists are used widely rather than matrix. Because the input graph is sparse, dictionary are always used to store graph as adjacent list. Global variable $graph\_pc$, $graph\_cp$ and $incoming$ are dictionary. Lists always store routes and edges information for local variable.

3.3. Performance

Following table show different performance with different dataset. 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

Solutions are accepted most all public dataset' test. But Linear Threshold model produce influence with large bias.

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] Kempe D, Kleinberg J, Tardos É. Maximizing the spread of influence through a social network[C]//Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003: 137-146.