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.

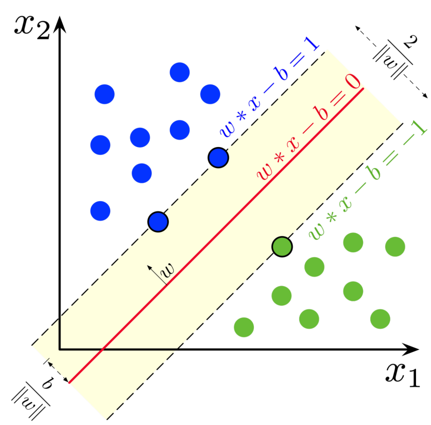

Figure 1. Separate data points into two part using SVM(from Larhmam)

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 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.

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

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. In order to reduce the time cost to update w with whole dataset, we always to choose part of dataset in random.

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

$$\frac{\partial L(w, b)}{\partial w} = -y_i x_i, if 1 - y_i((< w, x > + b) >= 0), otherwise 0$$

In above formula, α is the learning rate, which determine the moved distance in each update. In this project, learning rate always is updated during training, where learning rate being slower and slower.

$$w = w - \beta\alpha \frac{\partial L(w,b)}{\partial w}$$

In practical, we add one more direction at the end of each samplex_i. Final w combine original w and b together. So x is a $(m+1)n$ matrix, and w is a $m+1$ vector.

To avoid over fitting, k-fold cross validation is used. Whole training dataset is divided into 10 parts in equal in random. Each time, one part from 10 parts is chosen as test dataset, other 9 part as training dataset. Finally, the average of all w in 10 model is final w.

After training by SVM, w and b is final result, which can predict the label of test dataset.

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**: $(m+1)n$ dimensions numpy matrix to store input data.
- **y**: n dimensions numpy matrix of label of input data.
- **w**: $m+1$ dimensions 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
open input_file_name as file
create set data
for each line in file do
    split line, then package and append to data
end for
transform set data to matrix x
y ← x[:, −1]
x[:, −1] ← 1
initialize w in uniform distribution with length of m+1.

return x, y, w
```

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

Algorithm 2 cal_sgd

```
Input: x_i, y_i, w, α, β
Output: w_updated
    if y_i(x_i · w) < 1 then
        w_updated = w − β(αw − y_i x_i)
    else
        w_updated = w − βαyi
    end if
    return w_updated
```

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

Algorithm 3 train

```
Input: x, y, w, β, α
Output: w
    t ← 0
    while running time don't exceed ordered time do
        reorder x and x in the same time
        loss ← 0
        t ← t + 1
        β = 1/(αt)
        for each sample x_i and y_i in training data do
            w = cal_sgd(x_i, y_i, w, β, α)
        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 to test_data
    end for
    transform set test_data to matrix test_set
    append 1 at each end of column
    label_set ← sgn(test_set · w)
    return label_set
```

- **__main__**: main control flow

Algorithm 5 __main__

```
Input:
Output:
    pass through time_limit, train dataset, test dataset
    init
    use 10-hold cross validation train w
    use predict with test data set to predict final result
```

3. Empirical Verification

Empirical verification is verified offline with given dataset. K-fold cross-validation is used to verify and choose best result. In most of case, the correctness is above 90 percent.

3.1. Design

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

K-fold cross validation maybe reduce chance of overfitting, but it reduce the iteration of single gradient descent. Although, training without cross validation performance better, cross validation result still is suitable for unknown dataset.

Training time also determine the performance of final result, which seem that more time is flavor.

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.

Here are 10-fold cross validation.

Run Time(s)	Iteration	Correctness
10	10*66	95.3523
20	10*146	98.0510
30	10*226	97.8260
60	10*440	98.4258

Here are training without 10-fold cross validation.

Run Time(s)	Iteration	Correctness
10	670	97.2264
20	1356	98.3759
30	2064	97.9760
60	4176	98.8006

3.4. Result

Best model classify data set at 99.6 percent of correctness.

3.5. Analysis

During training, most time cost is iteration of update w. The cost of random gradient descent.

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 contributors, [Online], Available: https://en.wikipedia.org/wiki/Support_vector_machine, [Accessed: 31-Dec-2018].