Min-Max Classifier Refactoring

EI 328 Course Report Expurgated

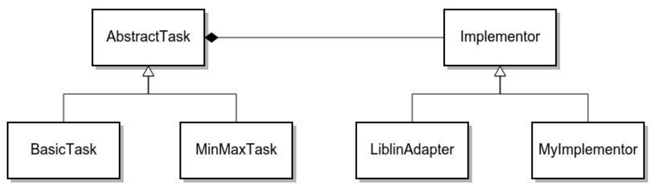
5120109159 左楠
Dept. of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, People's Republic of China
devinz1993@gmail.com

I. Introduction

This is the course project of EI328 in SJTU, which is delivered by Lv Baoliang, a professor who introduced the idea of min-max modular neural network in machine learning.

We are required to implement a 2-class classifier, which is applied in massive Japaneses patent classification. Data files data/train.txt and data/test.txt are provided by course TA, both of which have been well pre-processed. Our task is to implement the min-max modular neural network to solve this problem, based on both the library LIBLINEAR and a basic classifier we made in our own right.

The architecture of my project is as follow, which makes good use of the so-called *Bridge Pattern* and can easily adapt to the changes of both caller and callee side.



The class BasicTask simply calls the Implementor to tackle the problem, while MinMaxTask carries on the process of min-max modular neural network. MyImplementor is my self-made classifier based on logistic regression, and the class LiblinAdapter is an adapter between LIBLINEAR library and the interface of Implementor.

II. Min-Max Implementation

The static method *main_thread* generates multiple working threads and divides the training set into multiple sub-problems for them. When the training task is done, the main thread releases new semaphores, and working threads continue to serve as the MIN modules of the whole network. After that, the working threads terminate and the main thread will ultimately finish the MAX module and record the testing results.

The most challenging part of implementing the min-max modular neural network is how to divide the original training set into multiple well-balanced sub-problems. Here, I just briefly describe what I did.

To begin with, we shall divide the whole training set into a list of positive sample indexes and a list of negative sample indexes. In this problem, we also sort the two lists with respect to some prior knowledge of the patent classification result.

Then, given a group size parameter M, as well as the total number of positive sample N_1 and the total number of negative sample N_2, the numbers of positive groups and negative groups will be as follow:

$$n_1 = \left\lceil \frac{N_1}{M} \right\rceil, \quad n_2 = \left\lceil \frac{N_2}{M} \right\rceil.$$

We distribute the samples to the n_1 positive groups and n_2 groups based on the following equation:

$$N = \sum_{k=0}^{n-1} \lfloor (N+k)/n \rfloor.$$

Here, for positive and negative groups respectively, N should be N_1 and N_2, and n should be n_1 and n_2.

The upper bound of a group size is:

$$\left\lceil \frac{N}{\lceil N/M \rceil} \right\rceil < \frac{N}{\lceil N/M \rceil} + 1 \le M + 1$$

The lower bound of a group size is:

$$\left\lfloor \frac{N}{\lceil N/M \rceil} \right\rfloor > \frac{N}{\lceil N/M \rceil} - 1 > (\frac{1}{N} + \frac{1}{M})^{-1} - 1$$

Thus, suppose $N_1 \le N_0$, when $1 << M << N_0$, the ratio of positive and negative sample numbers in a sub-problem will be not less than:

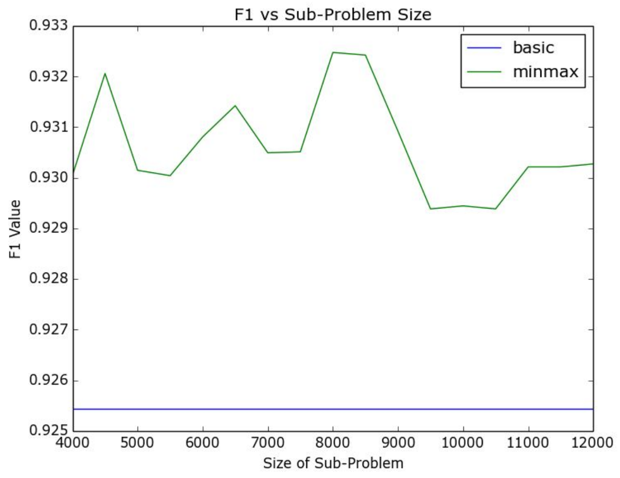
$$\frac{(N_0^{-1} + M^{-1})^{-1} - 1}{M+1} \approx \frac{M-1}{M+1} \approx 1$$

As you can see, the sub-problems thus created are approximately well-balanced.

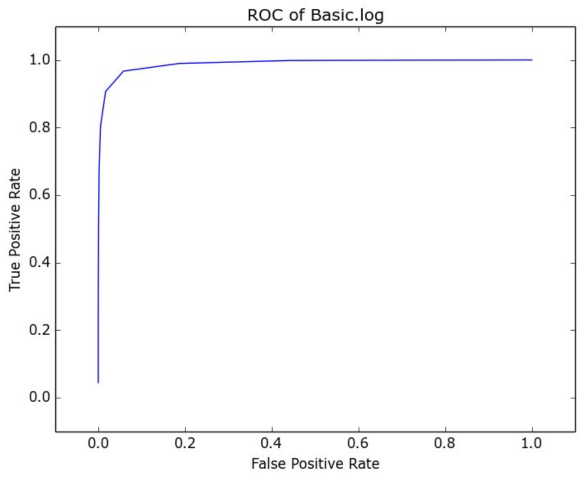
III. Accuracy Comparison

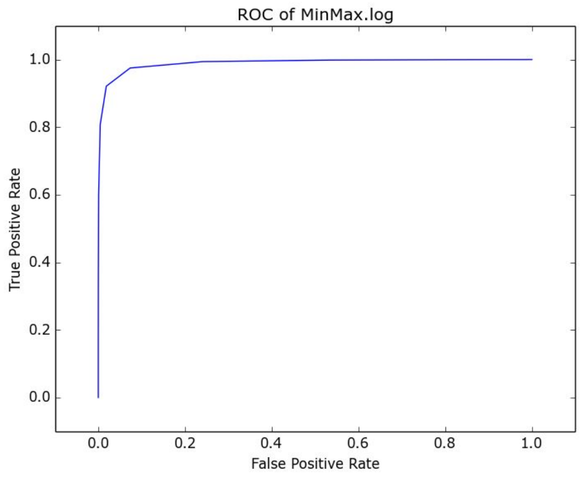
Here we compare the accuracy of different classifiers in terms of F1 values - an index that is more persuasive than the traditional accuracy when the learning problem is not balanced.

We determined the F1 values of program MinMaxTask with different group size parameters respectively, in comparison with that of program BasicTask, and finally got the following line chart.



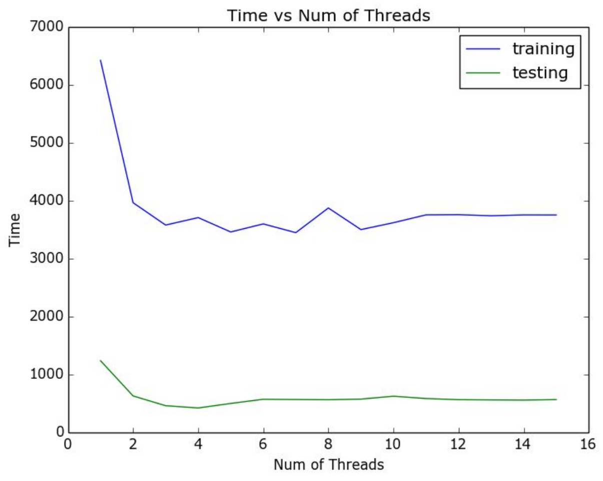
Moreover, we also plotted the ROC curves of both BasicTask and MinMaxTask, and here are the results:





IV. Time Performance

We compare the training and testing times of MinMaxTask with different number of working threads and plotted the following line char:



We noted that with two working threads MinMaxTask runs much faster than with only one working thread. However, as the number of working threads further increases, the time performance cannot get much better. This is probably limited by the hardware conditions - my personal lab-top has only four cores, and thus multiple threads cannot be fully paralleled and may also suffer a great amount of context switching.

V. Logistic Regression

The last part is about my self-made classifier. In BasicTask, its performance is as follow:

```
Thr = 0.0
    Training: 106787 ms elapsed.
    acc  = 0.9454824538188747
    F1   = 0.887628191141174
    TPR  = 0.8891803278688525
    FPR  = 0.03652744796759324
    Testing: 41 ms elapsed.
    *** CHANGE M ***
```

The following data is the performance of MinMaxTask:

```
N  =  7
M  =  8700
Thr = 0.0
    Training: 289355 ms elapsed.
    acc  = 0.9458794262425236
    F1   = 0.8870602529408517
    TPR  = 0.8777049180327869
    FPR  = 0.03233691856404526
    Testing: 556 ms elapsed.
    *** CHANGE M ***
    *** CHANGE N ***
```

My basic classifier is based on logistic regression, whose discriminant function is as follow:

$$y_n = (1 + \exp\{-w \cdot x_n\})^{-1}.$$

Then, we can easily get the following likelihood function:

$$L = \prod_{n=1}^{N} y_n^{t_n} (1 - y_n)^{1-t_n}$$

$$\Rightarrow -\ln L = \sum_{n=1}^{N} \{t_n \ln(1 + \exp\{-w \cdot x_n\}) + (1 - t_n) \ln(1 + \exp\{w \cdot x_n\})\}$$

$$\Rightarrow \nabla_w(-\ln L) = \sum_{n=1}^{N} (y_n - t_n) \cdot x_n$$

We use the traditional batch gradient descent method to approach the maximum point of the likelihood function, and as a result, the time performance leaves much to be desired.

This is the end of my expurgated lab report.