工科创4J

成员 沈键 / 朱文豪

你也可以通过扫描二维码在手机上观看

1. 概述

1.1 问题概述

本次要解决的是一个专利分类最顶层 (SECTION 层) 的两类分类问题，它主要有以下几个特征:

- 各类样本存在较为严重的不平衡
- 多标号现象普遍存在, 标号具有层次化结构
- 样本特征处于高维空间, 相对比较稀疏

1.2 实验环境

- 操作系统: Windows 10, Ubuntu 16.04
- 处理器: Intel Core i5 双核
- 内存: 4GB

1.3 分工

	1	2	3	4	5	6	7	R	S
沈键	√	√	√	√	√			√	√
朱文豪					√	√	√	√	√

2. 实验过程

2.1 直接使用 LIBLINEAR 学习

- 样本集/测试集预处理
- 训练 model
- 计算指标并绘制曲线

预处理

读取样本集/测试集，修改样本标号，使其符合标准格式[1/0 特征序号:特征值]，输出得到新的样本集/测试集。

```
function DataPro(txt,txt1)

ftxt = fopen(txt,'r+');
ftxt1= fopen(txt1,'w+');

while feof(ftxt)==0
    tline=fgetl(ftxt);
    S = regexp(tline, '\s+', 'split');
    s1 = char(S(1));
    if s1(1)=='A'
        S(1)=cellstr('1');
    else
        S(1)=cellstr('0');
    end

    len=length(S);
```

训练 MODEL

根据样本集调用 train 函数得到 model

```
%[trainY,trainX] = libsvmread('./train_new.txt');
%[testY, testX] = libsvmread('./test_new.txt');

load traindata
load testdata
model = train(trainY, trainX, '-c 1');
[maxlabel, accuracy, dec_values] = predict(testY, testX, model);
%save label_p1 maxlabel
```

计算指标并绘制曲线

- 在 predict 中根据不同的阈值判断分类结果,并统计其中 TP/TN/FP/FN 的数量,计算 p, F1 等指标。
- 循环修改预测的阈值, 输出 TPR-FPR, 作出 ROC 曲线，由梯形面积近似计算 AUC。

计算指标并绘制曲线

```
load traindata
load testdata

testSize=length(testY);
threshold=[-8,-4,-2,-1,-0.5,0,0.5,1,2,4,8];
thresnum=length(threshold);
TPRarr=zeros(thresnum,1);
FPRarr=zeros(thresnum,1);

for j=1:thresnum
    model = train(trainY, trainX, '-c 1');
    [~, ~, dec_values] = predict(testY, testX, model);
    maxlabel=zeros(testSize,1);
    maxlabel(dec_values>=threshold(j))=1;
    TP=0;
    FN=0;
```

2.2 使用 MIN-MAX 模块化网络

- 预处理 随机划分子问题
- 使用最小最大模块化网络训练
- 通过多处理器实现并行学习
- 计算指标并绘制曲线

划分子问题 - 随机

- 将样本随机打乱后按正负类分成两组, 在正类中分为 m 组, 在负类中分为 n 组, 一共生成 m*n 个 model
- 对于 m 和 n 的取值,我们做了一些选择(有 3/8, 4/4, 4/6 等)并比较最后结果发现差别不大, 最终采用了 4/4 来记录之后的数据。

采用MIN-MAX网络训练

- 对于每个测试样本,用每个子模块进行预测,对同一个 MIN 模块内所有子模块生成的预测值取最小, 作为该 MIN 模块的预测值。再对所有 MIN 模块的预测值取最大, 得到最终预测值。

采用MIN-MAX网络训练

```
predict_label = zeros(testSize,modelnum);
for i=1:modelnum
    [predict_label(:,i), ~, dec_values] = predict(testY, testX, mode
end
minlabel=ones(testSize,posnum);
for i=1:posnum
    beginc=negnum*(i-1)+1;
    endc=negnum*i;
    for k=beginc:endc
        minlabel(:,i)=predict_label(:,k)&minlabel(:,i);
    end
end
maxlabel=zeros(testSize,1);
for i=1:posnum
    maxlabel=maxlabel | minlabel(:,i);
end
```

通过多处理器实现并行学习

- 并行计算前需要先开启并行计算池，结束后关闭
- 通过 parfor 关键字将循环分解为独立的部分，由多个 worker 并行执行

通过多处理器实现并行学习

```
parpool('local',2);
tic
model = cell(modelnum,1);
for i=1:modelnum
    x=ceil(i/negnum);
    y=mod(i,negnum);
    if y==0
        y=negnum;
    end
    subY1=ones(posend(x)-posbegin(x)+1,1);
    subY2=zeros(negend(y)-negbegin(y)+1,1);
    subY=[subY1;subY2];
    subX1=pos(posbegin(x):posend(x),:);
    subX2=neg(negbegin(y):negend(y),:);
    subX=[subX1;subX2];
    model{i}=train(subY, subX, '-c 1');
```

2.3 结合先验知识分割样本

- 预处理 结合先验知识分割样本
- 使用最小最大模块化网络训练
- 通过多处理器实现并行学习
- 计算指标并绘制曲线

划分子问题 - 基于先验知识

- 处理训练集，将样本的第二层标号分离出来并保存在一个 txt 文档中
- 读入该文档，创建或修改结构体，根据其 index 数组组合相应的子问题的正/负样本
- 子问题的总数为 15*(36+19+8)=945 个,即一共需要训练 945 个 model

划分子问题 - 基于先验知识

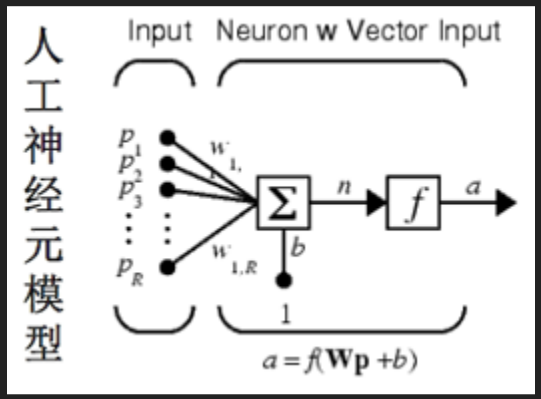
```
function classPro(txt,txt1)

ftxt = fopen(txt,'r+');
ftxt1= fopen(txt1,'w+');

while feof(ftxt)==0
    tline=fgetl(ftxt);
    S = regexp(tline, '\s+', 'split');
    s1 = char(S(1));
    classarr=regexp(s1, ',', 'split');
    classlen=length(classarr);
    wline=[];
    classtell={};
    for i=1:classlen
        flag=0;
        ch1=char(classarr(i));
```

2.4 实现 MLP 基分类器

- 预处理 随机分割样本
- 使用最小最大模块化网络训练

实现 MLP 基分类器

- MLP 采用 BP 算法，在实验中，我们采用了三层神经网络
- 若在规定学习轮数内得到的误差小于阈值,则停止学习,否则将在超出学习轮数后马上终止学习



实现 MLP 基分类器

```
void calculateOutput(vector<double> &input,vector<double> &output)
{
    int i,j;
    //calculate the output of hidden layer
    for (i=0;i<hiddenLayer->neuronNum;++i){
        double inputSum=0;
        //handle n dimensions
        for (j=0;j<hiddenLayer->neurons[i].inputNum-1;++j){
            inputSum+=hiddenLayer->neurons[i].weight[j]*input[j];
        }
        //handle n+1 dimension (bias)
        inputSum+=hiddenLayer->neurons[i].weight[j]*BIAS;
        //calcuate the activation
        hiddenLayer->neurons[i].activation=Sigmod(inputSum);
    }
    //calculate the output of output layer;
```

2.5 在集群上运行

- 部署 Spark 集群
- 样本集/测试集预处理
- 通过 Spark LIBLINEAR 实现并行学习

部署 SPARK 集群

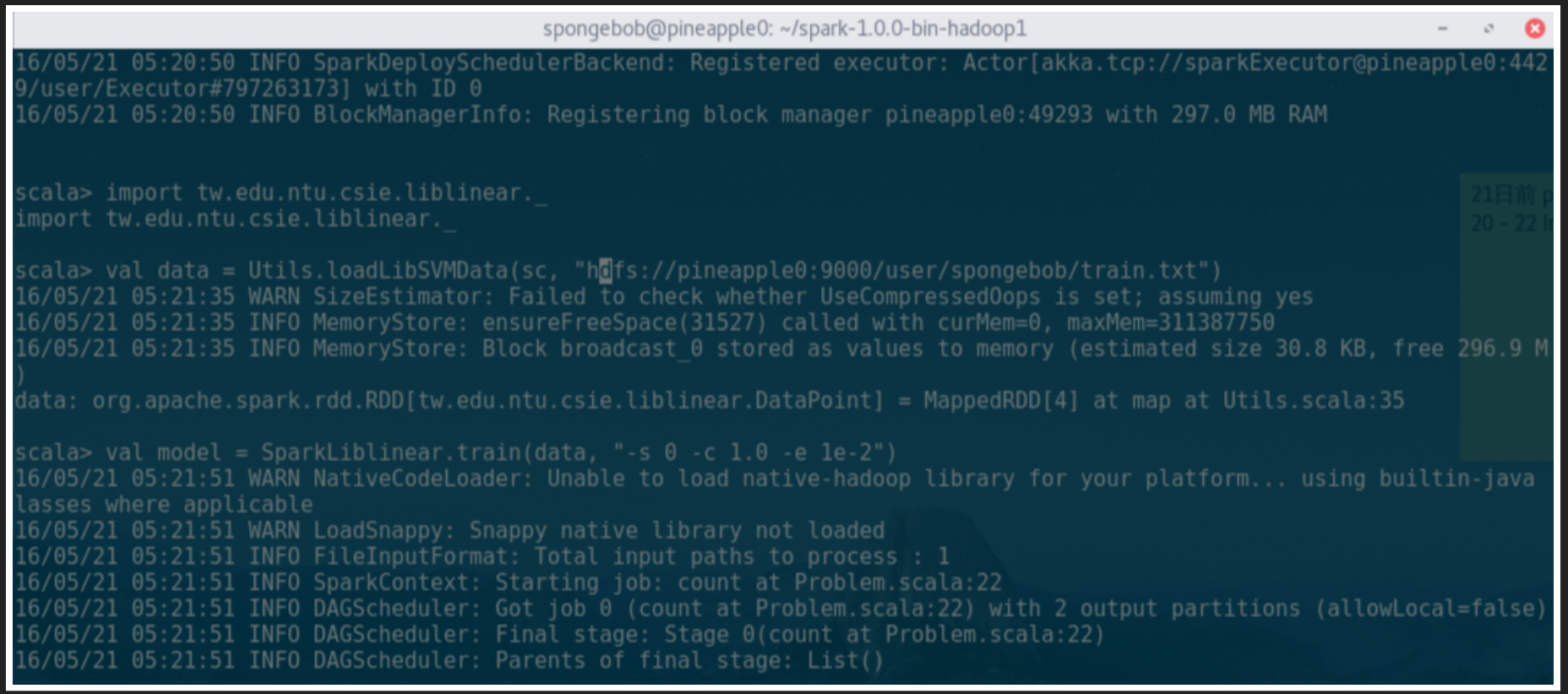
- 创建至少 3 个容器 或 virtualbox 镜像(本机)
- 配置每台机器的 hostname, 并修改 hosts （添加其他机器 hostname 和 ip）
- 选定一台机器作为 master, 添加其他机器为 slaves 到 /hadoop/conf/slaves 和 /spark/conf/slaves

通过 SPARK LIBLINEAR 学习

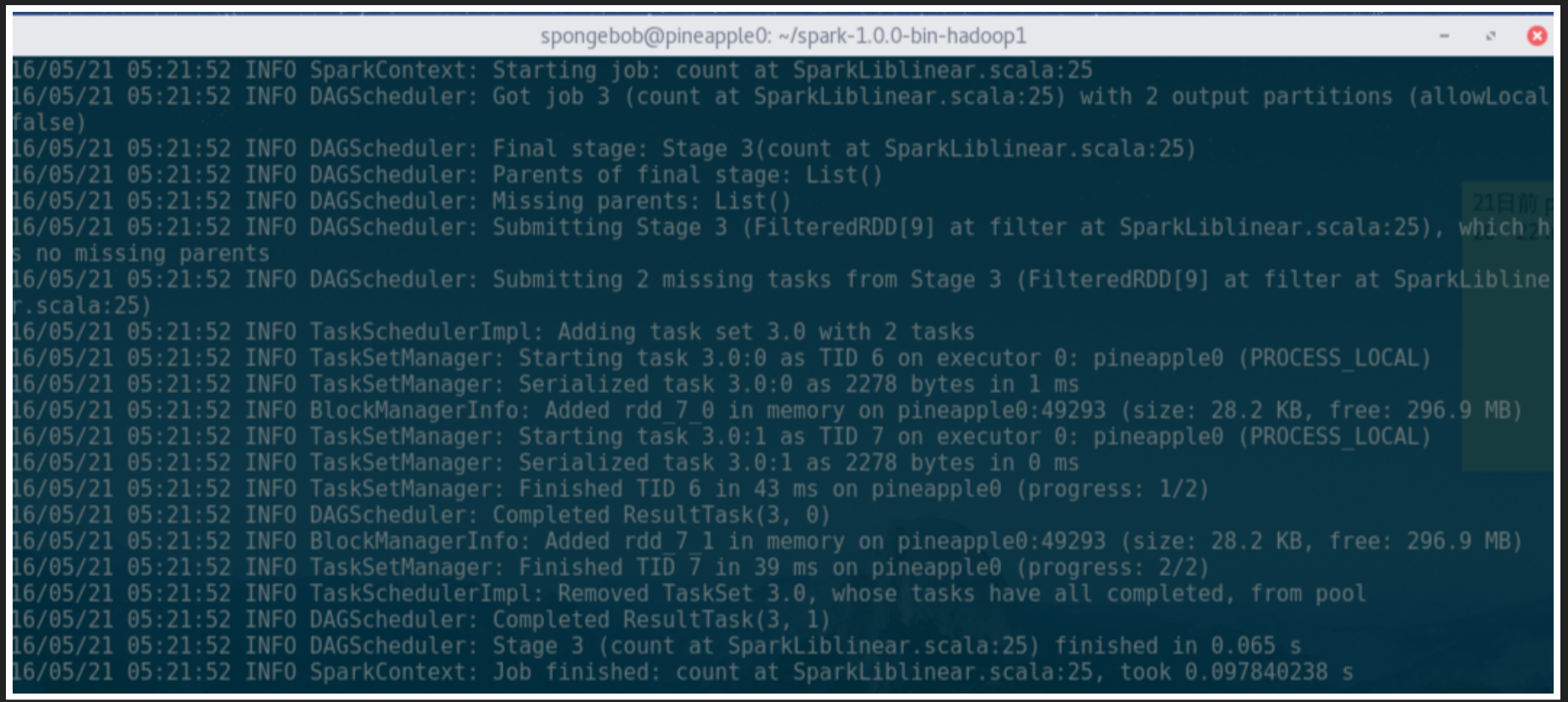
```
hadoop fs -put ~/train_new.txt train.txt
./spark-1.0.0-bin-hadoop1/sbin/start-all.sh
./spark-1.0.0-bin-hadoop1/bin/spark-shell --jars "/home/spongebob/sp
```

```
scala> import tw.edu.ntu.csie.liblinear._
scala> val data = Utils.loadLibSVMData(sc, "hdfs://pineapple0:9000/u
scala> val model = SparkLiblinear.train(data, "-c 1")
```

通过 SPARK LIBLINEAR 学习



通过 SPARK LIBLINEAR 学习



2.6 使用深度学习框架 CAFFE

- 样本集/测试集预处理
- 定义 solver 和 net
- 训练 model

预处理

- 由于样本特征是稀疏的,直接通过 svmread 读到的数据为 dict(字典)类型, 大部分特征为 0,需要将这些特征补全。
- 将 train 和 test 分别写入 h5 格式的文件

预处理

```
import numpy as np
import matplotlib.pyplot as plt
import os
os.chdir('..')

import sys
import h5py
import shutil
import tempfile

import sklearn
import sklearn.datasets
import sklearn.linear_model

import pandas as pd
```

定义 SOLVER 和 NET

- solver 借鉴了 caffe-master/examples/hdf5-classification/solver.prototxt, 只是对其 learning rate 以及迭代次数等做了少许修改。
- net 没有卷积层和池化层, 为简单的 2 个数据输入层、全连接层、softmax 层, 之后增加 ReLu (Rectified Linear Units) 层和全连接层, 使网络更深, 不断检验分类效果。

3. 实验结果

3.1 预测精度及相关性能

方法	子问题数	准确率	F1
liblinear direct train	1	96.49%	0.9261
liblinear random	16	96.49%	0.9261
liblinear priori knowledge	945	96.49%	0.9261
mlp	16	91.79%	0.8019
caffe	----	79.07%	----

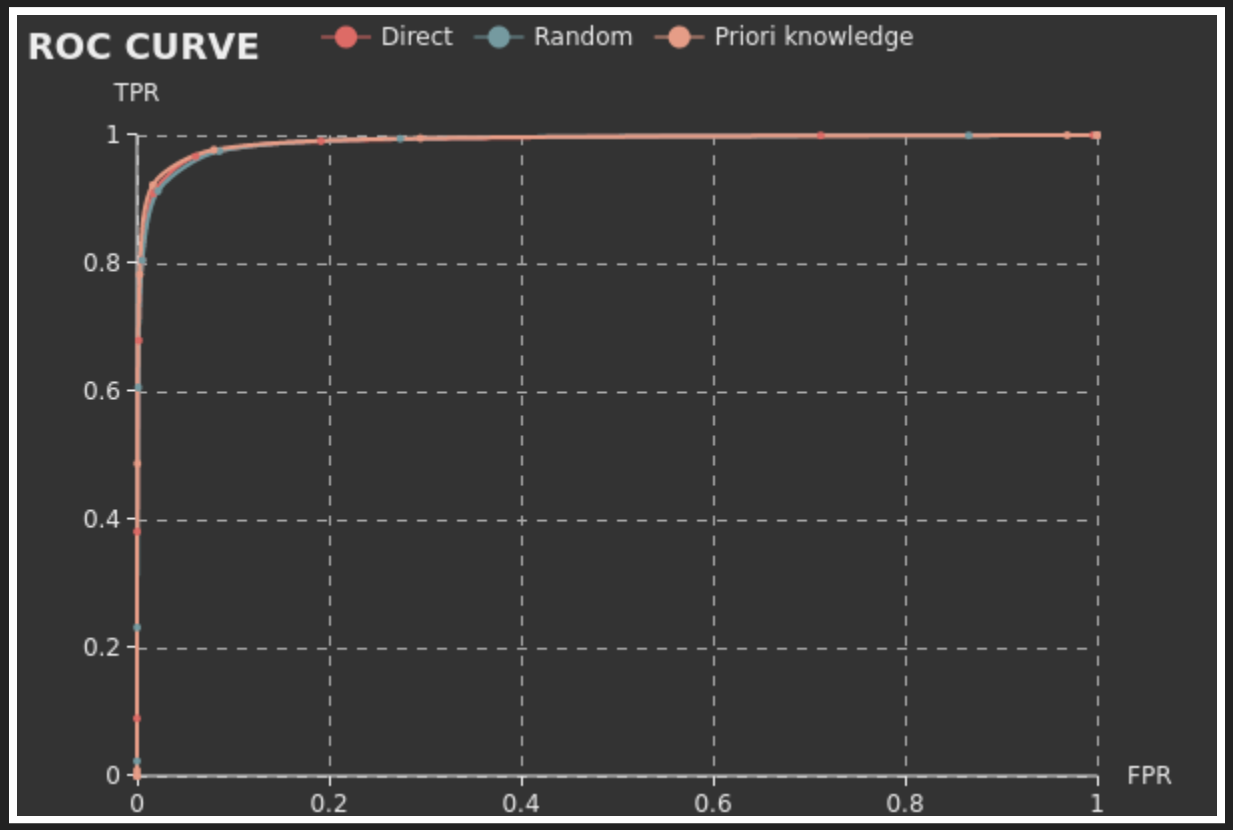
3.1 预测精度及相关性能

LIBLINEAR 直接学习即可以得到较高的精确度,进一步划分样本之后,受划分精度及样本子集之间平衡程度的影响, 分类的精度及 F1 值等性能并没有非常显著的提高。

3.2 F1 评价指标与 ROC 曲线

liblinear direct train			liblinear random minmax			liblinear priori knowledge		
threshold	fpr	tpr	threshold	fpr	tpr	threshold	fpr	tpr
-8	1	1	-8	1	1	-8	1	1
-4	0.996124	1	-4	0.999965	1	-4	1	1
-2	0.71232	0.999454	-2	0.866532	0.999672	-2	0.968885	1
-1	0.191577	0.990492	-1	0.274375	0.994536	-1	0.295293	0.995082
-0.5	0.061007	0.966776	-0.5	0.085766	0.975082	-0.5	0.080144	0.977158
0	0.017321	0.90918	0	0.02193	0.912787	0	0.016483	0.922186
0.5	0.005378	0.804918	0.5	0.005692	0.787322	0.5	0.002898	0.782186
1	0.001886	0.679563	1	0.001362	0.60623	1	0.000419	0.487322
2	0.00014	0.381202	2	0	0.231585	2	0	0.008743
4	0	0.089836	4	0	0.02306	4	0	0
8	0	0.000328	8	0	0	8	0	0
AUC	0.9883		AUC	0.9873		AUC	0.9889	

3.2 F1 评价指标与 ROC 曲线



3.2 F1 评价指标与 ROC 曲线

由本次实验的 F1 值和 ROC 曲线可知,三种方法取得的分类效果在伯仲之间,都十分好。而基于先验知识分解子问题后用 MIN-MAX 预测的方法相对来说比较出色。

3.3 串并行时间分析

方法	子问题数	训练时间/s	预测时间/s
随机分解串行	16	6.69	3.61
随机分解2处理器并行	16	12.23	5.18
随机分解串行	96	18	22.75
随机分解2处理器并行	96	17.32	17.7
先验知识分解串行	945	33.63	234.14
先验知识分解2处理器并行	945	17.04	160.81

3.3 串并行时间分析

通过观察，我们可以看出子问题数和串行时间的正相关关系，子问题数数越大，串行时间越长，这是符合我们预期的。并且我们发现在问题规模小的时候采取并行并没有取得减小运行时间的效果，这是因为本身使用多处理器并行就存在着额外开销，反而得不偿失。从并行加速比也可看出，如果不考虑数据传输等额外开销，通过服务器或者 GPU 实现全并行，可以达到极高的加速比。

参考文献

[1] Bao-Liang Lu, Masami Ito. Task Decomposition and Module Combinat
[2] Bao-Liang Lu, Hajime Kita, Yoshikazu Nishikawa. Multi-sieving ne
[3] Parallel learning of large-scale multi-label classification prob
[4] Roc: http://zh.wikipedia.org/wiki/ROC
[5] Large-scale Logistic Regression and Linear Support Vector Machin
[6] Caffe-logistic regression: http://nbviewer.jupyter.org/github/BV