# 超并行机器学习与海量数据挖掘

# 大作业报告

——EI328 工程实践与科技创新 IV-J

申继媛

5130309194

吴凡

5130309291

# 一、问题背景

## 1. 实验目的

大规模专利分类问题是一个十分有代表性的模式分类问题，它具有目前超大规模复杂模式分类问题得全部特征。本作业要解决的是一个专利分类最顶层[section]的两类分类问题，是大规模日文专利数据集的子集。本实验旨在通过对大规模专利分类问题的研究让同学们加深对并行机器学习、并行程序设计和大规模数据挖掘的理解，培养工程实践能力，了解已有方法和技术的局限性，为未来的理论研究和技术开发打下基础。

## 2. 实验描述

大规模专利分类问题属于多标号问题，在该实验中，我们对第一层标号 A, B, C 和 D 进行分类，且其中 A 类为正类，其他所有类为负类。训练所用数据集为在文件 train.txt，共含有 113128 个样本，其中 A 类 27876 个，B 类为 59597 个，C 类为 23072 个，D 类为 2583 个。测试用数据集为 test.txt，共含有 37786 个样本。

数据格式为每行代表一个样本，样本格式为：

标号 1 标号 2 标号 3……：数据

亦即，多标号情况下标号间以空格隔开，标号与数据间用冒号隔开（每个样本中的数据由标题、摘要、声明、正文四个部分按照顺序组成，每个部分以分号结尾，例 A01C/21/00，A01G/9/02 22：5.4721 74：23.4648 87：12.9582 101：6.2738 111：15.3368 149：2.9653）。所有样本的特征维数为 5000 维，并且已经对样本输入进行了归一化处理（取值在（0,1）之间）。

## 3. 实验要求

基础部分：

(1) 使用 LIBLINEAR 库直接学习上述两类问题，用常规的单线程程序实现；

(2) 用最小最大模块化网络解决上述两类问题,用随机方式分解原问题,每个子问题用 LIBLINEAR 学习,用多线程或多进程程序实现；

(3) 用最小最大模块化网络解决上述两类问题,用基于先验知识(层次化标号结构信息)的问题分解策略分解原问题,每个子问题用 LIBLINEAR 学习,用多线程或多进程程序实现；

(4) 给出上述三种分类器的 F1 值,画出这三种分类器的 ROC 曲线,并说明哪种分类器分类性能最好;

(5) 比较 LIBLINEAR 分类器与基于先验知识分解最小最大模块化 LIBLINEAR 分类器的训练时间和分类时间。最小最大模块化 LIBLINEAR 分类器只考虑完全并行情况下所需的时间,可以在代码中添加计时器对每个进程进行计时来确定完全并行情况下消耗的时间。

任选部分:

(6) 实现一个基分类器,如 MLP 或 SVM 多项式核,完成上述第 2 和第 3 个任务,并与 LIBLINEAR 作比较。

(7) 使用 GPU 或服务器实现上述第 1 至第 3 个任务。

## 二、方法导向

1、将本实验训练数据文件 train.txt、测试数据文件 test.txt 与 LIBLINEAR 库的自带样例 heart_scale 比较,发现数据格式不相同,即无法直接用 LIBLINEAR 对给定输入进行学习,这里我们采取的方法是先将给定两个输入文件转换成符合学习条件的训练数据文件和测试数据文件，然后直接用 LIBLINEAR 学习。

2、直接将给定输入数据按照（1，-1）分类成正集数据文件和负集数据文件，再进一步划分为更小的子文件并进行组合，得到组合后的小型训练数据文件。分别对得到的小型数据文件直接使用 LIBLINEAR 学习并预测,将所有小型预测值通过最小最大模块化网络求解得到最终的预测结果。

3、基于先验知识（层次化结构信息）：将数据文件 train.txt 按照分类的顺序进行排序（原始的 ABCD 序列排序变换成紧接其后的数字排序，具体的排序方法及思想见第四部分实验内容详述的预备处理部分），再通过与题 2 相同的最小最大模块化网络求解最终预测结果。

4、可以有两种实现方法：一是在程序运行的同时计算实时值并对性能文件进行格式化输出，再者可以在程序执行完成之后，将得到的预测数据与 test.txt 的标准进行比较得到对应每个时刻的瞬时值并同时对性能文件进行格式化输出。这里，我们采取的是后者，即将得到的文件逐一比较 TP、FP、TN、FN 值，并用公式输出 TPR 和 FPR 值做出 ROC 曲线。

5、添加计时器到程序中得出性能比较。

6、为了实现基分类器 SVM，需要类比基础部分实验的实现方法，参照 libsvm 的函数库代码，将其中不需要的功能和错误检测删去，还可以对文件读取进行一定的处理。注意到 svm 在处理小数据量的非线性问题上时性能较好，因而可以考虑划分成更小的数据文件进行训练操作。

# 三、实验环境搭建及运行说明

## 实验环境：

操作系统：VMWARE WORKSTATION 11.0 + UBUNTU 15.10

资源：LIBLINEAR-1.96、PTHREAD、EXCEL

程序语言：C(gcc compiler)

## 环境搭建：

1、VMWARE WORKSTATION 11.0 + UBUNTU 15.10

在官网 https://my.vmware.com/web/vmware/downloads 下载所要版本的 VMWARE WORKSTATION，直接按照指导安装，建议安装在空间充裕的盘。在官网 http://www.ubuntu.com/download/下载所要版本的 UBUNTU 镜像，并在 VMWARE 中添加镜像即可，这里不赘述。

2、LIBLINEAR-1.96

在网站 http://www.csie.ntu.edu.tw/~cjlin/liblinear/下载所要版本的 LIBLINEAR 包，并直接在终端 make，得到 train 和 predict 可执行文件。注意的是这两个可执行文件将拷贝到接下来的所有程序目录下用于学习及预测。

3、PTHREAD

在终端中执行两条安装指令$sudo apt-get install glibc-doc 和 $sudo apt-get install manpages-posix manpages-posix-dev 即可。

## 运行说明：

文件夹中已经编译好可执行文件，如果你有兴趣的话，也可以自己再编译一遍，运行说明如下：对于第一题，简单的$gcc -c p1.c $gcc -o p1 p1.o 即得到可执行文件；对于第二题和第三题，注意编译的时候要在后面添加 pthread 参数，即$gcc -o p2 p2.o -lpthread 即得到可执行文件。

# 四、实验内容详述之基础部分

## 1、预备处理

（1）基础部分第一题：

通过将自带 heart_scale 和 train.txt 对比，发现后者不能直接被 LIBLINEAR 学习预测的原因在于其数据格式的第一部分不是简单的(+1,-1)数字，因而我们需要将输入的 train.txt 文件修改为(1,-1)开始的格式，其中 A 用 1 代替，其他三类用-1 代替。我们想到两种可能的解决方法：一是修改原始的 train.c 和 predict.c 文件的读取输入部分，使得直接使用 LIBLINEAR 学习时也能够读取本实验的输入格式；二是简单地编写一个程序将 train.txt 改成标准的 LIBLINEAR 可读取格式（相应地也要修改 test.txt 文件）。

其一，调整 train.c 文件中的读取部分：

```
void read_problem (const char *filename)
{    ...
    for (i=0; i<prob.l; i++)
    {       ...
       if (label[0]=='A')
       {
            prob.y[i]=1;}
       else
       {
            Prob.y[i]=-1;}
    }  }
```

调整 predict.c 文件中的预测读取部分：

```
void    do_predict(FILE*input, FILE*output)
{...
    while (readline(input)!=NULL)
    {       ...
       if (label[0]=='A')
       {
            target_label=1;}
       else
       {
            target_label=-1;}
    }  }
```

其二，将输入文件改成标准格式：（例如，将‘A01C/21/00，A01G/9/02 22：5.4721 74：23.4648 87：12.9582 101：6.2738 111：15.3368 149：2.9653’转变为‘1 22：5.4721 74：23.4648 87：12.9582 101：6.2738 111：15.3368 149：2.9653’）

由以上两种方法都可以得到配对的（train、predict 可执行文件，train.txt，test.txt），进而就是两类问题，将在程序实现部分进一步说明。

（2）基础部分第二题

（不需要使用前面的程序将输入文件先转换成标准格式，因可以同时在分块正负集数据时进行转换。）不需要进行预备处理。直接在程序中读取原始数据进行训练和预测。

（3）基础部分第三题

需要对输入的文件先根据原本的标识信息进行结构层次排序。这里，我们理解的结构层次排序是将第二层的信息作为排序的主依据，进而提扩大数据的分布偶然性，提高模型准确性。即更详细地阐述如下：训练集原始输入数据文件的排序状况是 A/01;A/02;...;A/10;...;B/01;B/02;...;B/11;...;C/01;C/03;...;C12;D/01;D/02;...;D/13.在基于先验知识的结构层次排序之后得到的序列状况为 A/01;B/01;C/01;D/01;A/02;B/02;D/02;...这种排序的好处是防止训练数据过于块状集中，进而能提高学习效果。

## 2、程序实现

### 函数实现及说明

1）int main();

程序接口函数，在一题中直接调用函数训练预测；在二三题中创建多线程，调用其它函数训练预测。

2）void filter(char* filename);

读取训练集文件 train.txt，并将其每个样本按照标识分为正样本集和负样本集：若读取为 A 则分类+1，若不为 A 则分类-1。输出所有正样本集到文件 positive.txt，输出所有负样本集到文件 negative.txt。

3）void split_file(char* filename,int n);

将包含所有正样本集的文件 positive.txt 和包含所有负样本集的文件 negative.txt 进一步拆分为更小的文件 pos%d.txt 和 neg%d.txt。考虑

到输入数据正样本数与负样本数比例为27876:85252≈3.05，我们在后面的学习应该考虑划分比例。输入参数n整型用于指定数据被分成几部分。

4）void combine_p_n_sample();

将每个split_file产生的正样本集文件pos%d.txt和split_file产生的每个负样本集文件neg%d.txt进行组合，得到(#p)(#n)个自训练样本集tra%d.txt(注意特点是每#n个正序相邻tra文件有相同的正样本集)。

5) void* learning_thread(void* arg);

该函数直接调用LIBLINEAR中的‘train trainFile’指令对符合标准格式的训练数据文件进行学习，并输出训练模型到trainFile.model文件。

6）void* predicting_thread(void* arg);

该函数直接调用LIBLINEAR中的‘predict testFile trainFile.model resultFile’指令，在学习模型trainFile.model下预测testFile分类情况，并输出预测结果到resultFile中。这里第一题就直接为结果文件result.txt，第二三题则输出到中间结果文件out%d.txt。

7）void min();

该函数实现最小化操作（and逻辑），当所有输入文件的某样本预测均为1时，才输出预测值1否则输出-1。我实现的是把out%d.txt中所有拥有相同正样本集的文件进行最小化操作，输出结果到文件min%d.txt。

8）void max();

该函数实现最大化操作（or逻辑），当所有输入文件的某样本预测至少存在一个为1时，便输出预测值1否则输出-1。我实现的是把最小化操作后生成的所有min%d.txt进行最大化操作，输出到文件result.txt。

9）void performance_calculate();

该函数用于对比test.txt与result.txt文件，并统计预测性能。通过简单每次读取两个文件中的对应样本预测值比较，计算TP, FN, TN, FP, precision,recall,F1,TPR,FPR值，将它们和时间值输出，该函数运行后生成文件是performance.txt用于查看性能参数。

10）void clean();

由于该程序运行后会产生大量的中间文件，为了方便查看结果，可简单

移除文件 positive.txt，negative.txt，pos%d.txt，neg%d.txt，tra%d.txt，out%d.txt，tra%d.txt.model，min%d.txt，最后我们会剩下四个有用的四个文件 train.txt，test.txt，result.txt，performance.txt。

## 函数在程序中的应用

1）基础部分第一题：使用函数 learning_thread()，predicting_thread(),performance_calculate()；使用 clock 统计时间。该题直接学习训练文件，预测测试文件因而 main 函数执行两个函数分别直接在终端输入命令‘train train.txt’,‘predict test.txt train.txt.model result.txt’。可以在 performance.txt 中查看性能值。

2）基础部分第二三题：使用以上提及的所有函数以及 pthread 和 clock。首先使用 filter(“train.txt”);指令将 train.txt 文件分为正集文件 positive.txt 和负集文件 negative.txt。其次分别使用指令 split_file(“positive.txt”,positive_set_number)和 split_file(“negative.txt,negative_set_number);指令将正集文件分为#p 个小正集文件，将负集文件分为#n 个小负集文件。然后执行 combine_p_n_sample();指令将小数据文件一一组合成小训练文件。于是可以直接通过 pthread 创建相同数目的线程，每个线程处理一个小训练文件的学习，并合并这些线程。同样的，通过 pthread 再次创建相同数目的线程，每个线程处理一次预测，并合并这些预测。得到了基于每个小训练文件模型的(#p)(#n)个对 test.txt 的预测文件。将这些预测文件经过 min()函数进行最小模块化处理得到(#p)个中间文件，将这些中间文件经过 max()函数进行最大模块化处理得到最终 result.txt 预测结果文件。

3）基础部分第四题：这里输出的 F1 值是最终预测文件的性能值，直接在输出的 performance.txt 中查看。为了作出 ROC 曲线，采取简单循环将每一行样本的 TPR、FPR 值输出到一个文件，分别对前三题进行这样的处理，此过程之后将产生三个文件，对应前一二三题的预测 ROC 数据(已处理为 matlab 可读形式)，通过 matlab 软件直接读入三个二维数据点文件并作出完整的 ROC 曲线图。

4）基础部分第五题：直接查看各 performance.txt 中的时间值并对比。

# 五、实验结果及分析之基础部分

## 1、实验结果

(1) 直接使用 LIBLINEAR 训练预测 @ T1

| precision | recall | F1 |
|---|---|---|
| 0.943443 | 0.909727 | 0.926278 |
| TPR | FPR | Accuracy |
| 0.909727 | 0.017426 | 0.964934 |

(2) 随机分解原问题 @ T2

| precision | recall | F1 | TPR | FPR |
|---|---|---|---|---|
| 0.930855 | 0.911803 | 0.914652 | 0.911803 | 0.026191 |

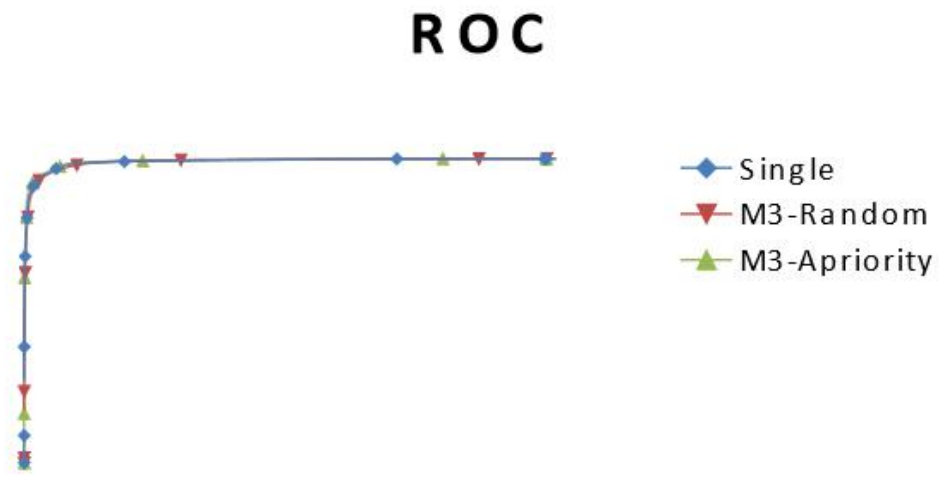
| Accuracy1 | Accuracy2 | Accuracy3 | Accuracy4 |
|---|---|---|---|
| 0.949055 | 0.947573 | 0.950934 | 0.948949 |
| Accuracy5 | Accuracy6 | Accuracy7 | Accuracy8 |
| 0.950352 | 0.950087 | 0.949479 | 0.950961 |
| Accuracy9 | Accuracy10 | Accuracy11 | Accuracy12 |
| 0.949532 | 0.949532 | 0.951146 | 0.950273 |
| Accuracy13 | Accuracy14 | Accuracy15 | Accuracy16 |
| 0.948843 | 0.950405 | 0.950273 | 0.950087 |
| Accuracy17 | Accuracy18 | Accuracy19 | Accuracy20 |
| 0.950511 | 0.948394 | 0.949611 | 0.949823 |
| Accuracy21 | Accuracy22 | Accuracy23 | Accuracy24 |
| 0.950484 | 0.950828 | 0.950431 | 0.948235 |

(3) 基于先验知识(层次化标号结构信息)分解原问题 @ T3

| precision | recall | F1 | TPR | FPR |
|---|---|---|---|---|
| 0.945257 | 0.912022 | 0.913120 | 0.912022 | 0.027343 |

| Accuracy1 | Accuracy2 | Accuracy3 | Accuracy4 |
|---|---|---|---|
| 0.95067 | 0.949981 | 0.950008 | 0.950034 |
| Accuracy5 | Accuracy6 | Accuracy7 | Accuracy8 |
| 0.949743 | 0.948658 | 0.948738 | 0.949479 |
| Accuracy9 | Accuracy10 | Accuracy11 | Accuracy12 |
| 0.949346 | 0.949188 | 0.949981 | 0.950511 |
| Accuracy13 | Accuracy14 | Accuracy15 | Accuracy16 |
| 0.950511 | 0.95014 | 0.949981 | 0.94887 |
| Accuracy17 | Accuracy18 | Accuracy19 | Accuracy20 |
| 0.950299 | 0.949876 | 0.949585 | 0.950458 |
| Accuracy21 | Accuracy22 | Accuracy23 | Accuracy24 |
| 0.951225 | 0.950564 | 0.949029 | 0.949743 |

(4) ROC 曲线图 @ T4




(5) 训练时间、分类事件统计 @ T5

| | 原始数据 | 随机排序 | 基于先验知识 |
|---|---|---|---|
| 训练时间 | 0.000962s | 0.000567s | 0.000051s |
| 分类时间 | 0.000760s | 0.000342s | 0.000057s |

## 2、结果分析

观察 ROC 曲线，三条曲线的差别不大，可知本实验所解决的问题本身的线性可分性比较好。可以直接对比表格中的数据结果，基于先验知识的问题分解，可以获得最好的结果及性能，而随机分解子问题再使用最小最大模块化方法并不会比直接对原问题进行训练获得更好的结果，在于本问题的实验数据本身的新型可分性使得直接在全部数据上进行训练和分类也能得到较好的准确率。反而使得在数据量小的子训练集上训练分类时的准确率不如直接训练分类的好，这也解释了第二题的准确率不如第一题高。由于本实验数据的特殊性，该实验结果并不具有普遍性。理论上对于真正的随机样本而言，使用最小最大模块化方法进行训练分类的效果会比直接对所有数据进行训练分类的效果好。

对比随机分解子问题和基于先验知识分解子问题，与课堂上提到的 spiral 例子相似，由于基于先验知识分解子问题使得数据集的分布更加具有普遍性，建立的模型更加准确，进而得到更好的分类效果。

在时间性能的比较上，实验数据比较理想，可见并行化的思想给训练分类的实现带来了有效的影响，在实验中将原数据集分解为 24 份之后，每个并行子问题的规模远远小于原问题的规模，使得时间性能得到了很大的提升。

# 六、实验内容详述之任选部分 6

## 1、理论分析

为了构造新的核函数,需要对核函数进行分析。

总体来讲,liblinear 使用的都是线形核。所谓的线形核，就是 K（x，xi）=x*xi 型的。更简单的讲，线形核的最终目的是寻找合适的参数（权重 w，通常为矩阵），使得对于任意的输入 x 在经过 w*x 的操作之后得出的值即为预测值。为了达到这个目的，训练过程即不断对 w*x 得到的预测值与实际值进行比对，并由 w*x 和实际值的距离来调整 w 以便缩小预测结果与实际值的差别。训练完成后得到的 w，我们相信此 w 下得 w*x 即为预测结果。

我们要使用多项式核，我们的目的即为找到 K（x，xi）=（a*x*xi+b）^p 型的核函数。和之前的线形核的原理类似，我们此时仍然需要合适的权重信息 w，但我们此时还需要 a 与 b（p 为先前固定的，且 p 为正整数，(a==1&b==0&p==1)

为假，否则即位线形核函数）。此时的 a 与 b 不一定要求是数字，也可能是矩阵，但为了训练的简单高效可以将 a 与 b 选定的常数。当然，我们的预测值=（a*x*输入值+b）^p。为了得到这样的核函数，训练过程即不断对（a*w*xi+b）^p 得到的预测值与实际值进行比对，并由（a*w*xi+b）^p 和实际值的距离来调整 w 以便缩小预测结果与实际值的差别。训练完成后，我们相信此（a*w*xi+b）^p 即为预测结果。但这只是但从理论来说，具体实现方面，不论是线形核函是多项式核都可以有许多的变化。

## 2、具体说明

将介绍 A coordinate descent algorithm for the dual of L2-regularized logistic regression problems 与我们作出的改变，以说明一个简单的多项式核可以在此问题中代替线形核方法:

**(1) 工具型函数**

已有的工具型函数：

sparse_operator::axpy(y[i]*alpha[2*i], xi, w) 作用为将样本点 xi 所有维度上的特征值反映到 w 的相应纬度,即: w= w+y[i]*alpha[2*i],*xi。

sparse_operator::dot(w, xi)作用为求出 w*xi 的值并返回此值,实际即对 xi 所有存在的特征维度上的值乘以 w 在相应纬度的值,之后将所有积相加。

sparse_operator::nrm2_sq(xi)作用为计算 xi 存在特征维度上值的平方。

现在的多项式核,需要新的工具来更新预测:

sparse_operator::axpyMLP(y[i]*alpha[2*i], xi, w, et, bia)

w= w+(y[i]-bia)/et*alpha[2*i],*xi

sparse_operator::dotMLP(w, xi, bia , et)=et*w*xi +bia

**(2) 输入参数**

已有的算法中：

void solve_l2r_lr_dual(const problem *prob, double *w, double eps, double Cp, double Cn)

多项式核函数需要新的参数：

void solve_l2r_lr_dual(const problem *prob, double *w, double eps, double et， double bia， double Cp, double Cn)

// et 即多项式中的 a，bia 即截距 b

如果 a 或 b 为矩阵形势，即每个纬度由自己的 a 或 b，这种参数输入输出的改变是必须的。但如果如果 a 与 b 为常数，实际上并不需要此改动，只要保证训练时的 a 与 b 与预测时的 a 与 b 保持一致即可。

**(3) 训练函数**

已有的对数据集的训练的整体迭代结构：

```
while (iter < max_iter)
{
……
for (s=0; s<l; s++)       //遍历所有的点，l 表示样本个数
{
while (inner_iter <= max_inner_iter)
{
if(fabs(gp) < innereps)
break;
……
}
}
……
if(Gmax < eps)
break;
if(newton_iter <= l/10)
//对最初的相当于输入样本数目十分之一的内层循环，不断调整内部迭代的阈值
innereps = max(innereps_min, 0.1*innereps);
}
```

即规定一个最大循环数，当低于限度值跳出循环。而每个循环需要遍历所有的样本点，对每个样本点都要调整使得预测函数值与真实值差距低于一定限度。而对前十分之一的输入样本，不断调整这个内部迭代的阈值，是一种基于数据特征的动态调整，可以加速收敛速度。

这样的结构基本能保证最终的收敛，且应用牛顿法调整内层循环的限度值可以在一定程度上促进更快的收敛。也就是说，这种结构基本是合理的,所以我们的核函数亦将使用此结构。

## 3、实现细节

### (1) 基本概念

| | |
|---|---|
| upper_bound[GETI(i)] | 样本 i 的界 |
| | (训练过程对于单个样本是靠近界的过程) |
| Cp | 正样本 |
| Cn | 负样本 |
| alpha[2*i] | 当前点的位置 |
| alpha[2*i+1] | 当前点到界的距离 |
| gp = a*(z-alpha_old) +sign*b+log(z/(C-z)) | 判断此样本是否已经正确预测的依据 |
| Gmax = max(Gmax, fabs(gp)) | 记录所有样本店中最坏的预测结果 |
| | (Gmax < eps 即可以说明所有点都能通过预测) |

### (2) 基本思想

● step1: 确定 alpha[2*i]移动方向

```
If (0.5*a*(alpha[ind2]-alpha[ind1])+b < 0)
{    ind1 = 2*i+1;
     ind2 = 2*i;
     sign = -1;   }
```

此处即为预测值与实际值的比对，其中 $b = y_w T_x = w \bullet x_i = alpha \bullet y_i \bullet x_i^2$

而因

$$0.5 \bullet a \bullet (alpha[ind2] - alpha[ind1])$$

$$= 0.5 \bullet x_i^2 \bullet (C - 2alpha)$$

$$= 0.5 \bullet x_i^2 \bullet C - x_i^2 \bullet alpha$$

故有 $0.5 \bullet a \bullet (alpha[ind2] - alpha[ind1]) + b = x_i^2 \bullet (alpha \bullet (y_i - 1) + 0.5 \bullet C)$

● step2: 判断 z 位于中界的左侧或右侧，这将决定关于 gp 函数的选取，

即当距离收敛状态较远则选择更大的步幅，接近收敛选取变化较慢的函数。

因 C - z < 0.5 * C 即 z > C/2,z 的更新方法决定 z<C，

*if(C - z < 0.5 * C)　　z = 0.1*z;*

可得：

◆ $z > C/2$

$$gp = a \bullet (z - alpha\_old) + sign \bullet b + \log(z/(C-z))$$

$$= x^2 \bullet (-0.9\, alpha\_old + sign \bullet alpha \bullet y_i) + \log(z/(C-z))$$

→ $sign = 1$:

$$alpha\_old = alpha$$

$$gp = x^2 * alpha\_old\,(y_i - 0.1) + \log(z/(C-z))$$

→ $sign = -1$:

$$alpha\_old = C - alpha$$

$$gp = x^2 * ((alpha\_old - C) \bullet y_i - 0.9 \bullet alpha\_old) + \log(z/(C-z))$$

◆ $z < C/2$

$$Z = alpha\_old$$

$$gp = sign \bullet b + \log(z/(C-z))$$

● step3: 每个样本点经过训练后的结果由 axpy(sign*(z-alpha_old)*yi,xi, w)记录至 w，即达到更新 w 的目的。

经过分析 w 与 y 的联系不影响 alpha，所以，我们不需要调整 alpha，只需要对 w 的更新调整即可达到目的。用(Yi - bia)/et 代替 Yi 即可得到正确的对应关系。

● step4: 对于预测函数,我们也需要改变以适应现在的模型。即:

将 dec_values[i] += w[(idx-1)*nr_w+i]*lx->value;

改为: dec_values[i] += et*w[(idx-1)*nr_w+i]*lx->value+bia 的形式

# 七、实验结果及分析之任选部分 6

## 1、实验结果

(1) 直接使用 LIBLINEAR 训练预测 @ T1

| precision | recall | F1 |
|---|---|---|
| 0.945899 | 0.888525 | 0.916314 |
| TPR | FPR | Accuracy |
| 0.888525 | 0.01238 | 0.960700 |

(2) 随机分解原问题 @ T2

| precision | recall | F1 | TPR | FPR |
|---|---|---|---|---|
| 0.893956 | 0.908415 | 0.901128 | 0.908415 | 0.034432 |

(3) 基于先验知识(层次化标号结构信息)分解原问题 @ T3

| precision | recall | F1 | TPR | FPR |
|---|---|---|---|---|
| 0.894142 | 0.907432 | 0.900738 | 0.907432 | 0.034327 |

## 2、结果对比分析

与基础部分使用 liblinear 的实验结果进行比较，实现的多项式核后的执行结果略有性能下降(基本一致)，均表现在准确度、精度、recall 值、F1 值、TPR 值和 FPR 值上。但是考虑到多项式核的运行时间开销太大，对于本题这种所给具有良好线性可分性的输入数据的情况下，使用线性核进行训练分类要比使用多项式核进行训练分类的效果更好。