HuoChengfu的博客
http://blog.sina.com.cn/hkingstar ［订阅］［手机订阅］

GBDT代码解读 (2013-05-19 19:16:20)

标签： gbdt 源码 机器学习 it 分类： 机器学习

亲，Treelink， boosting，决策树，GBDT，....，这些词都听过很多遍了吧。
了解算法细节吗？知道它们是程序怎么实现的吗？

阅读源码，了解算法细节，以便自己更加实际项目进行修改！
源码下载地址(CSDN)：http://download.csdn.net/detail/w28971023/4837775

Ready？ Begin

编译可执行文件
make clean; make

单步运行，解读程序
gdb gdb_train
r -r 0.8 -t 100 -s 0.03 -n 30 -d 5 -m test.model -f train
-r: random_feature_ratio
-t: infbox.tree_num
-s: infbox.shrink
-n: infbox.gbdt_min_node_size (阈值，如果节点内样本数小于这个值，则几点不再进行分裂)
-d: infbox.gbdt_max_depth
-m: model_filename
-f: train_filename (用于训练的数据文件，即训练数据)

代码解读
mian()
{
int res = read_conf_file(infbox, argc, argv); // 用默认值初始化配置参数
注释：
1. infbox.train_filename指定用于模型训练的文件，e.g. 每一条样本(一行)为 1 0:26503 1:511405 2:511405 3:500000 4:366 5:19.2954
2. infbox.fea_num表示训练文件中的特征数目(e.g. 6)，如上样本示例，特征从0开始计数
3. infbox.data_num表示样本数(e.g. 1500)，即infbox.train_filename文件的行数
4. infbox.rand_fea_num表示每棵树训练时，随机选择的特征数(e.g. 4)，每选择一个特征，计算一次分裂损失函数，对比得到最优的分裂特征
5. infbox.sample_num表示训练每一棵树需要的样本数(e.g. 1500)
6. infbox.gbdt_max_depth表示树的深度(e.g. 5)

read_train_file(x, y, infbox) // 读入训练文件
注释：
1. x = (double*) malloc (infbox.data_num * infbox.fea_num * sizeof(double)); // 样本数*特征数， 训练样本特征

博客等级：17
博客积分：1648
博客访问：178,301
关注人气：325
获赠金笔：9
赠出金笔：0
荣誉徽章：

相关博文

韩国夜店里的惹火美女
Hi-Korea


```
2. y = (double*) malloc (infbox.data_num * sizeof(double)); // 样本数，存储样本标注值
{
   while(getline(fptrain,line))  // ifstream fptrain(infbox.train_filename); 即为训练数据文件
   {
      cnt = 0;
      count = splitline(line, items, infbox.fea_num+5, ' '); // 以空格切割样本的每个特征，返回特征数 (e.g. 8)
      y[cnt] = value; // 训练目标值，cnt表示样本计数下标
      x[cnt*infbox.fea_num + fid] = value; // 每个样本(cnt)的特征(fid)对应的特征值(value)，没有则是默认值NO_VALUE
      cnt++;
    }
}

gbdt_model_t* gbdt_model = gbdt_regression_train(x, y, infbox);  // 模型训练
{
   //gbdt_model表示训练得到的最终的模型，其具体由gbdt_model->reg_forest指代
   gbdt_model_t* gbdt_model = (gbdt_model_t*)calloc(1, sizeof(gbdt_model_t));
   gbdt_model->info = infbox;
   gbdt_model->feature_average = (double*)calloc(gbdt_model->info.fea_num, sizeof(double));  // 每个feature算一个均值
   gbdt_model->reg_forest = (gbdt_tree_t**) calloc(gbdt_model->info.tree_num, sizeof(gbdt_tree_t*)) // 森林，树的数目

   // 计算gbdt_model->feature_average，每个特征的均值，对于x中的有些样本不含有某个特征，则赋予均值
   fill_novalue_feature(x_fea_value, gbdt_model->info.fea_num, gbdt_model->info.data_num, gbdt_model->feature_average)


   for (int i=0; i< infbox.data_num; i++)
   {
      y_gradient[i] = y_result_score[i]; // 标注的目标值
      y_pred[i] = 0;  // 训练过程中的预测值
   }

   // 训练过程中用到的变量， e.g. infbox.sample_num=1500
   data_set->fea_pool     = (int *) calloc(infbox.fea_num, sizeof(int));
   data_set->fvalue_list  = (double *) calloc(infbox.sample_num, sizeof(double));
   data_set->y_list       = (double *) calloc(infbox.sample_num, sizeof(double));
   data_set->fv           = (double *) calloc(infbox.sample_num, sizeof(double));
   data_set->order_i      = (int *) calloc(infbox.sample_num, sizeof(int));
   // e.g. nrnodes=3001，pgbdtree表示模型中的一棵树
   pgbdtree->nodestatus   = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->depth        = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->ndstart      = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->ndcount      = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->lson         = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->rson         = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->splitid      = (int*) calloc (nrnodes, sizeof(int));
   pgbdtree->splitvalue   = (double*) calloc (nrnodes, sizeof(double));
   pgbdtree->ndavg        = (double*) calloc (nrnodes, sizeof(double));
   pgbdtree->nodesize     = 0;

   // training iteration， the main part
   for(int j = 0; j < infbox.tree_num; ++j)
   {
      for (int i = 0; i< infbox.sample_num; i++)  // 每棵树训练所用到的样本数e.g. 1500
      {
         index[i] = i;  // 节点开始的样本下表，index中是1,2,...，对应到样本空间的下标可能是乱序的
      }

      for (int i = 0; i < infbox.data_num ; i++)  // 所有样本数
      {
         sample_in[i] = i;
      }
```

```
        pgbdtree->nodesize = 0; // clear pgbtree

        // build a single regression tree
        gbdt_single_tree_estimation(x_fea_value, y_gradient, infbox, data_set,
index, pgbdtree, nrnodes)
        int gbdt_single_tree_estimation(double *x_fea_value,double *y_gradient,gbdt_info_t
gbdt_inf,bufset* data_set,int* index,gbdt_tree_t* gbdt_single_tree,int nrnodes)
        {
           spinf->bestid = -1; // 最优分隔特征id
           for (int i = 0; i < gbdt_inf.sample_num; ++i)   index[i] = i;  // 初始赋值，训练本棵树
所用的的样本下表，  1,2,...,1499
           int ncur = 0;
           // 根节点初始化
           gbdt_single_tree->nodestatus[0] = GBDT_TOSPLIT;
           gbdt_single_tree->ndstart[0]    = 0;
           gbdt_single_tree->ndcount[0]    = gbdt_inf.sample_num;
           gbdt_single_tree->depth[0]             = 0;
           for (int i = 0; i < gbdt_inf.sample_num; ++i)   //计算训练样本目标值的均值
              avg = (i * avg + y_gradient[index[i]]) / (i + 1);
           gbdt_single_tree->ndavg[0] = avg;

           // 如果当前(根)节点的样本数少于节点分裂要求的样本数，则分解终止
           if (gbdt_single_tree->ndcount[0] <= gbdt_inf.gbdt_min_node_size)
           {
              gbdt_single_tree->nodestatus[0] = GBDT_TERMINAL;
              gbdt_single_tree->lson[0]             = 0;
              ......
              gbdt_single_tree->nodesize            = 1;
              return 0;
           }

           // start main loop, nrnodes是训练当前树的样本数的2倍+1，e.g. 3001
           for(int k=0; k
           {
              // initialize for next call to findbestsplit
             nodeinfo ninf;
             ninf.index_b = gbdt_single_tree->ndstart[k]; // begin,起始样本下标
             ninf.index_e = gbdt_single_tree->ndstart[k] + gbdt_single_tree->ndcount[k] - 1;
// end ,结束
             ninf.nodenum = gbdt_single_tree->ndcount[k]; // 节点样本数
             ninf.nodesum = gbdt_single_tree->ndcount[k] * gbdt_single_tree->ndavg[k];
             ninf.critparent = (ninf.nodesum * ninf.nodesum) / ninf.nodenum;  // 节点均方

             // 决策树分裂，随机选择rand_fea_num个特征尝试对某一节点分裂
             // 貌似是以分裂后的均方误差最小化作为评判依据
             jstat = gbdt_tree_node_split(gbdt_inf, data_set, x_fea_value,
y_gradient, ninf, index, spinf)
             int gbdt_tree_node_split(gbdt_info_t gbdt_inf, bufset* data_set, double
*x_fea_value, double *y_score, nodeinfo ninf, int* index, splitinfo* spinf)
              {
                 spinf->bestsplit = -1;   //当前节点选定的特征分裂值
                 spinf->bestid = -1;      // 当前节点选定的分裂特征

                 for (int i=0; i < gbdt_inf.fea_num; ++i) data_set->fea_pool[i] = i; // 初始化特
征池

                 critmax = 0;  // 用于选定最优的特征
                 for (int i = 0; i < gbdt_inf.rand_fea_num; ++i)  // 随机选择rand_fea_num个特
征进行分裂，并确定最优的一个分裂特征
                 {
                    int select = rand() % (last+1);  // 随机选择一个特征
                    int fid = data_set->fea_pool[select];
                    data_set->fea_pool[select] = data_set->fea_pool[last];
                    data_set->fea_pool[last] = fid; last--;
```

```
                    for (int j = ninf.index_b; j <= ninf.index_e; j++)
                    {
                        data_set->fvalue_list[j] = x_fea_value[index[j]* gbdt_inf.fea_num + fid];
// 读取所有样本该维特征的值
                        data_set->fv[j] = data_set->fvalue_list[j];
                        data_set->y_list[j] = y_score[index[j]];
                    }

                    for (int j = 0; j < gbdt_inf.sample_num; ++j)
                    {
                        data_set->order_i[j] = j;   // 当前树所用到的样本下标
                    }

                    R_qsort_I(data_set->fv, data_set->order_i, ninf.index_b+1 ,
ninf.index_e+1);
                    void R_qsort_I(double *v, int *I, int i, int j)
                    {
                        double R = 0.375;  // 设置默认值，用于选择作为比较标准的样本点
                        ii = i;   m = 1;

                        if (i < j)
                        {
                            if (R < 0.5898437) R += 0.0390625; else R -= 0.21875;
                            k = i;
                            ij = i + (int)((j - i)*R);   // 计算选定的标准样本点下标

                            // 通过比较进行交换，标准样本之前的样本小于其，之后的大于其
                            vt = v[ij];
                            if (v[i] > vt)  I[ij] = I[i]; I[i] = vt; it = I[ij];

                            l = j;
                            if (v[j] < vt)
                            {
                                v[ij] = v[j]; v[j] = vt; vt = v[ij];
                                if (v[i] > vt)
                                {
                                    v[ij] = v[i]; v[i] = vt; vt = v[ij];
                                }
                            }

                            for(;;) // vt分割，之前的小于其，忠厚的大于其
                            {
                                l--;  for(;v[l]>vt;l--);

                                vtt = v[l];
                                k=k+1; for(;v[k]
                                if (k > l) break;
                                v[l] = v[k]; v[k] = vtt;
                            }

                             m++;
                             if (l - i <= j - k) {}
                             else {}
                        }
                        else {}   //(i > j)

                    } // end of  R_qsort_I(double *v, int *I, int i, int j)， 实现按选定特征值的升
序排序

                    // 选取当前特征的最优分裂值
                    double left_sum = 0.0;    int left_num = 0;
                    double right_sum = ninf.nodesum;    int right_num = ninf.nodenum;
                    for (int j=ninf.index_b; j< ninf.index_e; j++)
                    {
                        d = data_set->y_list[data_set->order_i[j]];  // 样本下标也对应上述特征升
序排列发生了变化, e.g. j=0, order_i[j]=48
                        left_sum  += d;    right_sum -= d;
                        left_num++;  right_num--;
```

```
                    if (data_set->fv[j] < data_set->fv[j+1]) // 依次比较相邻的两个样本（已排序），找到最优的分裂值
                    {
                        crit = (left_sum * left_sum / left_num) + (right_sum * right_sum / right_num) - ninf.critparent;
                        if (crit > critvar) {tmpsplit = (data_set->fv[j] + data_set->fv[j+1]) / 2.0; critvar = crit;}
                    }
                } // end of for (int j=ninf.index_b; j< ninf.index_e; j++)

                if (critvar > critmax) // 选择从开始到当前循环，最好的分裂特征及分裂值
                { spinf->bestsplit = tmpsplit; spinf->bestid = fid; critmax = critvar;}

            } // end of for (int i = 0; i < gbdt_inf.rand_fea_num; ++i)

            if( spinf->bestid >= 0)
            {
                int nl = ninf.index_b;
                for (int j= ninf.index_b; j<= ninf.index_e; j++)  // 调整左子节点的样本下标
                {
                    if (x_fea_value[index[j]* gbdt_inf.fea_num + spinf->bestid] <= spinf->bestsplit)
                    {
                        data_set->order_i[nl] = index[j]; //update data->set
                        nl++;
                    }
                }

                for (int j= ninf.index_b; j<= ninf.index_e; j++) {...} // 调整右子节点的样本下标

                // 将调整后的样本赋值到index，进行下一次训练
                for (int j= ninf.index_b; j<= ninf.index_e; j++) {index[j] = data_set->order_i[j];}

                spinf->pivot = nl;  // 存储当前分裂节点，e.g. 785
                return 0;
            }
            else {return 1}

        } // end of bdt_tree_node_split(gbdt_inf, data_set, x_fea_value, y_gradient, ninf, index, spinf)

        gbdt_single_tree->splitid[k] = spinf->bestid; // 当前节点的分裂特征，e.g. k=0(根节点)
        gbdt_single_tree->splitvalue[k] = spinf->bestsplit;  // 特征分裂值，e.g. 344112
        gbdt_single_tree->nodestatus[k] = GBDT_INTERIOR;

        // 左子节点开始、结束样本下标，右子节点开始、结束样本下标，深度+1
        gbdt_single_tree->ndstart[ncur + 1] = ninf.index_b; ...
        gbdt_single_tree->ndcount[ncur + 1] = spinf->pivot - ninf.index_b;  ...
        gbdt_single_tree->depth[ncur + 1] = gbdt_single_tree->depth[k] + 1; ...

        // 计算左右子节点的sum和square
        for (int j = ninf.index_b; j < spinf->pivot; ++j)
        {
            d = y_gradient[index[j]]; m = j - ninf.index_b; avg = (m * avg + d) / (m+1);
        }
        for (int j = ninf.index_b; j < spinf->pivot; ++j)
        {
            var += (y_gradient[index[j]] - avg) * (y_gradient[index[j]] - avg);
        }
        var /= (spinf->pivot - ninf.index_b);
        .....

        gbdt_single_tree->lson[k] = ncur +1;
        gbdt_single_tree->rson[k] = ncur +2;
```

```
                ncur += 2;

            } // end of  for(int k=0; k

            gbdt_single_tree->nodesize = ncur+1;

        } // end of gbdt_single_tree_estimation, for one tree

        for (int i=0; i // 在模型里面存储当前生成的这棵树
        {
            gbdt_model->reg_forest[j]->nodestatus[i] = pgbdtree->nodestatus[i];
            gbdt_model->reg_forest[j]->ndstart[i] = pgbdtree->ndstart[i];
            gbdt_model->reg_forest[j]->splitid[i] = pgbdtree->splitid[i];
            ....
        }

        for (int i=0; i< infbox.data_num; i++)
        {
            for (int k=0; k
            {
                x_test[k] = x_fea_value[i * infbox.fea_num + k];
            }

            // 预测当前训练样本
            gbdt_tree_predict(x_test, gbdt_model->reg_forest[j], y_pred[i],
infbox.shrink);
            int gbdt_tree_predict(double *x_test, gbdt_tree_t *gbdt_single_tree, double& ypred,
double shrink)
            {
                int k = 0;
                while (gbdt_single_tree->nodestatus[k] != GBDT_TERMINAL)
                {
                    if (x_test[m] <= gbdt_single_tree->splitvalue[k])
                        k = gbdt_single_tree->lson[k];
                    else
                        k = gbdt_single_tree->rson[k];
                }
                ypred += shrink * gbdt_single_tree->ndavg[k];  // 预测结果
                return 0;
            } // end of gbdt_tree_predict(double *x_test, gbdt_tree_t *gbdt_single_tree,
double& ypred, double shrink)

            y_gradient[i] = y_result_score[i] - y_pred[i];  // 更新梯度
        } // end of for (int i=0; i< infbox.data_num; i++)

    } // end of or(int j = 0; j < infbox.tree_num; ++j), for all trees
} // end of gbdt_model_t* gbdt_model = gbdt_regression_train(x, y, infbox), for the model

// 存储模型参数
gbdt_save_model(gbdt_model, infbox.model_filename)
int gbdt_save_model(gbdt_model_t* gbdt_model, char* model_filename)
{
    FILE* model_fp = fopen(model_filename, "wb");
    save_gbdt_info(gbdt_model->info, model_fp);
    gbdt_save_reg_forest(model_fp, gbdt_model->reg_forest, gbdt_model->info);
    fwrite(gbdt_model->feature_average, sizeof(double), gbdt_model->info.fea_num,
model_fp);

}


} // end of main
```

本文是基于程序的一个框架，感兴趣的同学同时可以看看这篇文章：

http://blog.csdn.net/w28971023/article/details/8249108

4 4

喜欢 赠金笔

分享：

阅读(15423) | 评论 (5) | 收藏(0) | 转载(6) | 喜欢▼ | 打印 | 举报 已投稿到： 排行榜

前一篇：C++效率笔记两三点
后一篇：高斯混合模型

评论 重要提示：警惕虚假中奖信息 [发评论]

幻影离杀

lz，求GBDT的源码，我没有积分，所以不能下载，不知lz那里有没有源码？

2013-10-17 17:28 回复(0)

随风而逝

博主源代码是哪个公司的版本

2013-10-23 16:43 回复(1)

nwfengwolf

LZ，可以上传百度云盘一份么？CSDN没有积分，谢谢！

2014-8-15 16:25 回复(0)

-Hello_World

代码 地址http://t.cn/RhC8389

2014-9-20 21:35 来自 -Hello_World 的评论 回复(0)

xiaoguangY

为什么说index是1,2,3...对应的样本是乱序的
好像每棵树index都是1,2,3... 不应该是随机生成的吗？

2015-1-16 17:09 回复(0)

发评论

登录名： 密码： 找回密码 注册 记住登录状态

分享到微博 评论并转载此博文

>>> 拖动滑块完成拼图验证 >>>

发评论

〈 前一篇
C++效率笔记两三点

后一篇 〉
高斯混合模型