多广告位融合训练详细设计

目前不同广告位的学习任务是独立进行的，为了在不同广告位的数据间共享信息，考虑用多任务学习的思想将多个广告位的学习任务按层次结构统一成一个大的学习问题。

为了优化新的目标函数，考虑采用ADMM的框架将训练数据按广告位拆分成多个子任务，每个子任务用FTRL训练，这样可以保证在第一次迭代的时候模型等价于目前的分广告位训练[2]。

1 Multi-task Learning

我们仍然采用LR模型，但是假设模型参数w_t可以做如下分解：

$$w_t = w_0 + v_t$$

同时假设w_0和v_t都服从0均值的高斯分布：

$$\begin{aligned} w_0 &\sim \mathcal{N}(0, \lambda_w I) \\ v_t &\sim \mathcal{N}(0, \lambda_v I) \end{aligned}$$

假设任务t的训练样本为$\{(x_1^t, y_1^t), \cdots, (x_{n_t}^t, y_{n_t}^t)\}$，联合学习任务的优化目标为：

$$\sum_{t=1}^{T} \sum_{i=1}^{n_t} \mathcal{L}((w_0 + v_t)^T x_i^t) + \lambda_w \|w_0\|_1 + \lambda_v \|v_t\|_1$$

新的损失函数里w_0起到了连接不同子任务的作用，λ_w控制了连接的强度，λ_w越大连接强度越强，当$\lambda_w = 0$时模型等价于分广告位训练。

2 ADMM

我们把问题改写成ADMM[1]的形式：

$$\begin{aligned} \min_{w_0 \ldots w_t, v_1 \ldots v_t} \quad & \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}((w_0 + v_t)^T x_i^t) + \lambda_w \|w_0\|_1 + \lambda_v \sum_{t=1}^{T} \|v_t\|_1 \\ s.t. \quad & w_t = w_0 \, (t = 1 \ldots T) \end{aligned}$$

增广拉格朗日函数为：

$$\sum_{t=1}^{T} \frac{1}{n_t} \mathcal{L}(\boldsymbol{y}_t, X_t(w_t + v_t)) + \lambda_w \|w_0\|_1 + \lambda_v \sum_{t=1}^{T} \|v_t\|_1 + \sum_{t=1}^{T} \alpha_t^T (w_t - w_0) + \frac{\rho}{2} \sum_{t=1}^{T} \|w_t - w_0\|_2^2$$

迭代步骤为：

1. $v_t^{k+1} = \operatorname{argmin} \frac{1}{n_t} \mathcal{L}(\boldsymbol{y}_t, X_t(w_t^k + v_t)) + \lambda_v \|v_t\|_1$

2. $$\begin{aligned} w_t^{k+1} &= \operatorname{argmin} \tfrac{1}{n_t} \mathcal{L}\big(\boldsymbol{y}_t, X_t\big(w_t + v_t^{k+1}\big)\big) + (\alpha_t^k)^T w_t + \tfrac{\rho}{2} \|w_t - w_0^k\|_2^2 \\ &= \tfrac{1}{n_t} \mathcal{L}\big(\boldsymbol{y}_t, X_t\big(w_t + v_t^{k+1}\big)\big) + \tfrac{\rho}{2} \Big\| w_t - w_0^k + \tfrac{1}{\rho} \alpha_t^k \Big\|_2^2 \end{aligned}$$

3. $w_0^{k+1} = \operatorname{argmin} \lambda_w \|w_0\|_1 - \sum_{t=1}^{T} (\alpha_t^k)^T w_0 + \frac{\rho}{2} \sum_{t=1}^{T} \left\| w_t^{k+1} - w_0 \right\|_2^2$

$$=\begin{cases}\frac{1}{T}(\sum_t (w_t^{k+1}+\alpha_t^k/\rho)-\lambda_w/\rho) \text{ if } \sum_t (w_t^{k+1}+\alpha_t^k/\rho)>\lambda_w/\rho \\ \frac{1}{T}(\sum_t (w_t^{k+1}+\alpha_t^k/\rho)+\lambda_w/\rho) \text{ if } \sum_t (w_t^{k+1}+\alpha_t^k/\rho)<-\lambda_w/\rho \\ 0 \text{ else}\end{cases}$$

4. $\alpha_t^{k+1}=\alpha_t^k+\rho\left(w_t^{k+1}-w_0^{k+1}\right)$

其中：

1. 如果我们将w_t的初值设为0，第1步就是一个L_1正则的逻辑回归，如果用FTRL训练就得到了目前分广告位训练的模型；
2. 第2步是L_2正则的逻辑回归，第1步和第2步可以多个广告位并行训练；
3. 第3步是Soft-thresholding；
4. 第4步是对偶变量α的梯度上升；

3 ADMM模块

ADMM算法作为一种分布式算法，计算节点分为worker和master两种节点，其中worker节点用于分广告位子任务的训练，主要更新子任务的v_t、w_t和拉格朗日系数α_t，master节点只有一个，用于更新全局权重向量w_0。

ADMM计算步骤如下：

算法 ADMM

1. 初始化各子任务对应的v_t、w_t和α_t，步长$\boldsymbol{\rho}$，全局权重向量w_0，λ_w和λ_v
2. For each iteration
   3. 每个worker更新v_t
   4. 每个worker更新w_t
   5. Allreduce：所有v_t和w_t reduce到 mater
   6. master更新w_0
   7. Broadcast：分发w_0到每个worker
   8. 每个worker更新α_t

每个worker需要对v_t、w_t和α_t这三个参数做更新计算。

v_t更新	$v_t \leftarrow \underset{v}{\text{argmin}} \frac{1}{n_t}\mathcal{L}(\boldsymbol{y}_t, X_t(w_t+v))+\lambda_v\|v\|_1$
	输入：子任务t的训练样本集$(X_t, \boldsymbol{y}_t):=\{(x_1^t, y_1^t), ..., (x_{n_t}^t, y_{n_t}^t)\}$，以及$w_t$
	输出：v_t

w_t更新	$w_t \leftarrow \underset{w}{\text{argmin}} \frac{1}{n_t}\mathcal{L}(\boldsymbol{y}_t, X_t(w+v_t))+\frac{\rho}{2}\left\|w-w_0+\frac{1}{\rho}\alpha_t\right\|_2^2$
	输入：子任务t的训练样本集$(X_t, \boldsymbol{y}_t):=\{(x_1^t, y_1^t), ..., (x_{n_t}^t, y_{n_t}^t)\}$，$v_t$,$\alpha_t$以及$w_0$
	输出：w_t

α_t更新	$\alpha_t \leftarrow \ \alpha_t+\rho(w_t-w_0)$
	输入：w_0, α_t 以及w_t
	输出：α_t

master需要对w_0 进行更新

w_0更新	$w_0 \leftarrow \begin{cases} \frac{1}{T}(\sum_t (w_t+\alpha_t/\rho)-\lambda_w/\rho) \text{ if } \sum_t (w_t+\alpha_t/\rho)>\lambda_w/\rho \\ \frac{1}{T}(\sum_t (w_t+\alpha_t/\rho)+\lambda_w/\rho) \text{ if } \sum_t (w_t+\alpha_t/\rho)<\text{-}\lambda_w/\rho \\ 0 \text{ else} \end{cases}$
	输入：$\rho, \lambda_w, \{w_t\}$和$\{\alpha_t\}$
	输出：w_0

4 ADMM实现接口

AdmmConfig类

用于保存admm各参数配置的类,包含训练步长以及全局权重向量等参数

AdmmConfig类的公有成员变量：

`step_size`	浮点类型，步长ρ
`global_var`	浮点类型，参数λ_w
`bias_var`	浮点类型，参数λ_v
`global_weights`	vector类型，全局权重向量w_0
dim	整型， 权重向量的维数

FtrlConfig类

用于保存FTRL各参数配置的类,包含训练步长以及全局权重向量等参数

FtrlConfig类的公有成员变量：

l_1	浮点类型，L1正则系数λ_1
`l_2`	浮点类型，L2正则系数λ_2
`alpha`	浮点类型，参数α
`beta`	浮点类型，参数$\boldsymbol{\beta}$
dim	整型， 权重向量的维数

构造函数和初始化函数：

1. `FtrlConfig(const AdmmConfig& admm_params);`
   FTRL配置参数初始化
   输入：AdmmConfig配置参数

Worker 类

用于实现worker节点所做的子任务训练步骤，主要包括对子任务t的 v_t、w_t和拉格朗日系数α_t更新函数，Worker类设有成员变量保存子任务t的v_t、w_t和拉格朗日系数α_t以及子任务的ID。

Worker类的成员变量：

`base_vec_`	vector类型，参数w_t
`bias_vec_`	vector类型，参数v
`langr_vec_`	vector类型，参数α_t
`psid_`	string类型， psid广告位id
`num_part_`	int类型， 加载数据的分块数
`partid_`	int 类型， 当前数据块的序号

Worker类的成员函数：

1. `void BaseUpdate(SampleSet& train_set,SampleSet& test_set,const AdmmConfig& admm_params);`
   参数w_t的更新函数
   输入：训练集 train_set，验证集test_set，以及配置参数admm_params

2. `void BiasUpdate(SampleSet& train_set,SampleSet& test_set,const AdmmConfig& admm_params);`
   参数v_t的更新函数
   输入：训练集 sample_set，验证集test_set，以及配置参数admm_params

3. `void LangrangeUpdate(const AdmmConfig& admm_params);`
   拉格朗日系数α_t的更新函数
   输入：样本参数 sample_set，以及配置参数admm_params

4. `std::vector<real_t> GetWeights(AdmmConfig& admm_params,std::vector<real_t>& ptr) const;`
   返回最终求解的$w_t + \alpha_t/\rho$ 向量，结果保存在ptr数组中

Master 类

用于实现master节点所做的全局权重向量w_0计算，仅包含一个对w_0更新的成员函数。

Master类的成员函数：

1. `bool GlobalUpdate(const vector<vector<real_t>>& workers, AdmmConfig& admm_params, int num_work`
   参数w_0的更新函数
   输入：参数workers保存了所有worker节点的$\{w_t + \alpha_t/\rho\}$，以及配置参数admm_params，wokers的数量num_worker
   其中workers的每个元素表示一个worker节点提交的$w_t + \alpha_t/\rho$向量

注：

每个worker节点都需向master提交其对应的$w_t + \alpha_t/\rho$向量，此外因为admm_params的各成员变量在worker训练阶段保持不变，所以只需其中一个worker节点向master提交一次admm_params。

master向各worker节点广播admm_params参数。

FtrlSolver 类

用于实现FTRL算法求解带条件约束的LR模型。 Worker类的成员函数BaseUpdate和BiasUpdate都是通过调用FtrlSolver的对象来实现参数w_t和v_t的求解：通过类AdmmConfig的对象间接初始化类FtrlSolver对象，并设置截距向量，然后为FtrlSolver的Run成员函数传入样本集和截距向量完成模型训练得到解向量。

FtrlSolver类提供的初始化和调用接口函数：

1. `void Init(FtrlConfig &params);`
   FtrlSolver的初始化函数
   输入：ftrl配置参数params

2. `void Run(SampleSet& train_set, SampleSet& test_set ,`
   `const std::vector<real_t>& offset,const std::vector<real_t>& reg_offset);`
   运行FTRL训练样本集
   输入：训练集train_set，测试集test_set，目标函数的截距向量offset，L2正则项的截距向量reg_offset

Metric 类

实现了 AUC和LogLoss的计算模块

Metric类提供的AUC和LogLoss函数接口：

1. `real_t Auc(SampleSet& sample_set,std::vector<std::vector<real_t>>& weights,bool T);`
   auc计算模块，返回auc值
   输入：样本集 sample_set，权重向量组weights，布尔变量T用于控制输出样本信息（训练集/验证集）

2. `real_t LogLoss(SampleSet& sample_set,std::vector<std::vector<real_t>>& weights, bool T);`
   logloss计算模块，返回样本集上的 logloss值
   输入：样本集 sample_set，权重向量组weights，布尔变量T用于控制输出样本信息（训练集/验证集）

参考文献

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1--122, 2011.

[2] Theodoros Evgeniou and Massimiliano Pontil. Regularized multi--task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109--117. ACM, 2004.