Alink漫谈(十二)： 在线学习算法FTRL 之 整体设计

目录

- Alink漫谈(十二)： 在线学习算法FTRL 之 整体设计
  - 0x00 摘要
  - 0x01概念
    - 1.1 逻辑回归
      - 1.1.1 推导过程
      - 1.1.2 求解
      - 1.1.3 随机梯度下降
    - 1.2 LR的并行计算
    - 1.3 传统机器学习
    - 1.4 在线学习
    - 1.5 FTRL
      - 1.5.1 regret & sparsity
      - 1.5.2 FTRL的伪代码
      - 1.5.3 简要理解
  - 0x02 示例代码
  - 0x03 问题
  - 0x04 总体逻辑
  - 0xFF 参考

0x00 摘要

Alink 是阿里巴巴基于实时计算引擎 Flink 研发的新一代机器学习算法平台，是业界首个同时支持批式算法、流式算法的机器学习平台。本文和下文将介绍在线学习算法FTRL在Alink中是如何实现的，希望对大家有所帮助。

0x01概念

因为 Alink 实现的是 LR + FTRL，所以我们需要从逻辑回归 LR 开始介绍。

1.1 逻辑回归

Logistic Regression 虽然被称为回归，但其实际上是分类模型，并常用于二分类。Logistic 回归的本质是：假设数据服从这个分布，然后使用极大似然估计做参数的估计。

逻辑回归的思路是，先拟合决策边界(不局限于线性，还可以是多项式)，再建立这个边界与分类的概率联系，从而得到了二分类情况下的概率。

1.1.1 推导过程

我们从线性回归开始说起。某些情况下，使用线性的函数来拟合规律后取阈值的办法是行不通的。行不通的原因在于拟合的函数太直，离群值(也叫异常值)对结果的影响过大。但是我们的整体思路是没有错的，错的是用太"直"的拟合函数，如果我们用来拟合的函数是非线性的，不这么直，是不是就好一些呢?

所以我们下面来做两件事:

- 找到一个办法解决掉回归的函数严重受离群值影响的办法.
- 选定一个阈值.

对于第一件事，我们用sigmod函数把**回归函数掰弯**。

对于二分类问题，1表示正例，0表示负例。逻辑回归是在线性函数输出预测实际值的基础上，寻找一个假设函数函数h_θ(x) = g(θ，x)，将实际值映射到到0，1之间。逻辑回归中选择对数几率函数（logistic function）作为激活函数，对数几率函数是Sigmoid函数（形状为S的函数）的重要代表。

对于第二件事，我们选定阈值是0.5。

意思就是，当我选阈值为0.5，那么小于0.5的一定是负例，哪怕他是0.49。此时我们判断一个样本为负例一定是准确的吗? 其实不一定，因为它还是有49%的概率为正例的。但是，即便它是正例的概率为0.1，则我们随机选择1w个样本来做预测，还是会有接近100个预测它是负例结果它实际是正例的误差。无论怎么选，误差都是存在的，所以我们选定阈值就是在选择可以接受误差的程度。

1.1.2 求解

到这里，逻辑回归的由来我们就基本理清楚了，我们知道逻辑回归的判别函数就是

$$h(z) = \frac{1}{1+e^{-x}}, z = W^T X$$

$$h(z) = \frac{1}{1+e^{-x}}, z = W^T X$$

如何求解逻辑回归？也就是如何找到一组可以让 h(z) 全都预测正确概率最大的W。

求解逻辑回归的方法有非常多，我们这里主要聊下梯度下降和牛顿法。优化的主要目标是找到一个方向，参数朝这个方向移动之后使得损失函数的值能够减小，这个方向往往由一阶偏导或者二阶偏导各种组合求得。

梯度下降是通过 J(w) 对 w 的一阶导数来找下降方向，并且以迭代的方式来更新参数。

牛顿法的基本思路是，在现有极小点估计值的附近对 J(w) 做二阶泰勒展开，进而找到极小点的下一个估计值。

1.1.3 随机梯度下降

当样本数据里N很大的时候，通常采用的是随机梯度下降法，算法如下所示：

```
while {
    for i in range(0,m):
        w_j = w_j + a * g_j
}
```

随机梯度下降的好处是可以实现分布式并行化，具体计算流程是：

1. 在每次迭代的时候，随机抽样一定比例的样本作为当前迭代的计算样本。
2. 对计算样本中的每一个样本，分别计算不同特征的计算梯度。
3. 通过聚合函数，对所有计算样本的特征的梯度进行累加，得到每一个特征的累积梯度以及损失。
4. 最后根据最新的梯度以及之前的参数，对参数进行更新。
5. 根据更新的参数计算损失函数误差值，如果损失函数误差值达到允许的范围，那么停止迭代，否则重复步骤1

1.2 LR的并行计算

从逻辑回归的求解方法中我们可以发现这些算法都是需要计算梯度的，因此逻辑回归的并行化最主要的就是对目标函数梯度计算的并行化。

我们看到目标函数的梯度向量计算中只需要进行向量间的点乘和相加，可以很容易将每个迭代过程拆分成相互独立的计算步骤，由不同的节点进行独立计算，然后归并计算结果。

所以并行 LR 实际上就是在求解损失函数最优解的过程中，针对寻找损失函数下降方向中的梯度方向计算作了并行化处理，而在利用梯度确定下降方向的过程中也可以采用并行化。

如果将样本矩阵按行划分，将样本特征向量分布到不同的计算节点，由各计算节点完成自己所负责样本的点乘与求和计算，然后将计算结果进行归并，则实现了“ 按行 并行的LR”。

按行并行的LR解决了样本数量的问题，但是实际情况中会存在针对高维特征向量进行逻辑回归的场景（如广告系统中的特征维度高达上亿），仅仅按行进行并行处理，无法满足这类场景的需求，因此还需要 按列 将高维的特征向量拆分成若干小的向量进行求解。

1.3 传统机器学习

传统的机器学习开发流程基本是以下步骤：

1. 数据融合，获取训练以及评估数据集。
2. 特征工程。
3. 构建模型，比如LR，FM等。
4. 训练模型，获得最优解。
5. 评估模型效果。
6. 保存模型，并在线上使用训练的有效模型进行训练。

这种方式主要存在两个瓶颈：

1. 模型更新周期慢，不能有效反映线上的变化，最快小时级别，一般是天级别甚至周级别。
2. 模型参数少，预测的效果差；模型参数多线上predict的时候需要内存大，QPS无法保证。

比如，传统Batch算法中每次迭代对全体训练数据集进行计算（例如计算全局梯度），**优点是精度和收敛还可以，缺点是无法有效处理大数据集（此时全局梯度计算代价太大），且没法应用于数据流做在线学习。**

针对这些问题，一般而言有两种解决方式：

- 对1采用On-line-learning的算法。
- 对2采用一些优化的方法，在保证精度的前提下，尽量获取稀疏解，从而降低模型参数的数量。

1.4 在线学习

在线学习（ OnlineLearningOnlineLearning ）代表了一系列机器学习算法，特点是每来一个样本就能训练，能够根据线上反馈数据，实时快速地进行模型调整，使得模型及时反映线上的变化，提高线上预测的准确率。

传统的训练方法在模型训练上线后，一般是静态的，不会与线上的状况有任何的互动，加入预测错误，只能在下一次更新的时候完成修正，但是这个更新的时间一般比较长。

Online Learning训练方法不同，会根据线上的预测结果动态调整模型，加入模型预测错误，从而及时做出修正，因此Online Learning能够更加及时地反应线上变化。

Online Learning的优化目标是使得整体的损失函数最小化，它需要快速求解目标函数的最优解。

在线学习算法的特点是：每来一个训练样本，就用该样本产生的loss和梯度对模型迭代一次，一个一个数据地进行训练，因此可以处理大数据量训练和在线训练。常用的有在线梯度下降（OGD）和随机梯度下降（SGD）等，本质思想是对上面【问题描述】中的**未加和的单个数据的loss函数 L（w，zi）做梯度下降，**因为每一步的方向并不是全局最优的，所以整体呈现出来的会是一个看似随机的下降路线。

1.5 FTRL

FTR是FTRL的前身，思想是每次找到让之前所有样本的损失函数之和最小的参数。

FTRL，即 Follow The Regularized Leader，是在之前的几个工作上产生的，主要出发点就是为了提高稀疏度且满足精度要求。FTRL 在FTL的优化目标的基础上，加入了正则化，防止过拟合。

FTRL的损失函数一般也不容易求解，这种情况下，一般需要找一个代理的损失函数。

代理损失函数需要满足以下条件：

1. 代理损失函数比较容易求解，最好是有解析解。
2. 代理损失函数求得的解，和原函数的解的差距越小越好

为了衡量条件2中的两个解的差距，引入regret的概念。

1.5.1 regret & sparsity

一般对于在线学习来说，我们致力于解决两个问题： **降低 regret 和提高 sparsity**。其中 regret 的定义为：

$$Regret = \sum_{t=1}^{T} l_t(w_t) - min_w \sum_{t=1}^{T} l_t(w)$$

$$Regret = \sum_{t=1}^{T} l_t(w_t) - min_w \sum_{t=1}^{T} l_t(w)$$

其中 t 表示总共 T 轮中的第 t 轮迭代，ℓt 表示损失函数，w 表示要学习的参数。Regret 表示 "代理函数求出来的解" 离 "真正损失函数求出来的解" 的损失差距。

第二项 表示得到了所有样本后损失函数的最优解，因为在线学习一次只能根据少数几个样本更新参数，随机性较大，所以需要一种稳健的优化方式，而 regret 字面意思是 “后悔度”，意即更新完不后悔。

在理论上可以证明，如果一个在线学习算法可以保证其 regret 是 t 的次线性函数，则：

$$\lim_{t\to\infty} \frac{Regret(t)}{t} = 0$$

$$\lim_{t\to\infty} \frac{Regret(t)}{t} = 0$$

那么随着训练样本的增多，在线学习出来的模型无限接近于最优模型。即随着训练样本的增加，代理损失函数和原损失函数求出来的参数的实际损失值差距越来越小。而毫不意外的，FTRL 正是满足这一特性。

另一方面，现实中对于 sparsity，也就是模型的稀疏性也很看重。上亿的特征并不鲜见，模型越复杂，需要的存储、时间资源也随之升高，而稀疏的模型会大大减少预测时的内存和复杂度。另外稀疏的模型相对可解释性也较好，这也正是通常所说的 L1 正则化的优点。

1.5.2 FTRL的伪代码

Algorithm 1 Per-Coordinate FTRL-Proximal with L_1 and L_2 Regularization for Logistic Regression

#With per-coordinate learning rates of Eq. (2).
Input: parameters α, β, λ_1, λ_2
$(\forall i \in \{1, \dots, d\})$, initialize $z_i = 0$ and $n_i = 0$
for $t = 1$ **to** T **do**
  Receive feature vector $\mathbf{x}_t$ and let $I = \{i \mid x_i \neq 0\}$
  For $i \in I$ compute

$$w_{t,i} = \begin{cases} 0 & \text{if } |z_i| \leq \lambda_1 \\ -\left(\frac{\beta+\sqrt{n_i}}{\alpha} + \lambda_2\right)^{-1}(z_i - \text{sgn}(z_i)\lambda_1) & \text{otherwise.} \end{cases}$$

  Predict $p_t = \sigma(\mathbf{x}_t \cdot \mathbf{w})$ using the $w_{t,i}$ computed above
  Observe label $y_t \in \{0, 1\}$
  for all $i \in I$ **do**
    $g_i = (p_t - y_t)x_i$ *#gradient of loss w.r.t. w_i*
    $\sigma_i = \frac{1}{\alpha}\left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right)$ *#equals $\frac{1}{\eta_{t,i}} - \frac{1}{\eta_{t-1,i}}$*
    $z_i \leftarrow z_i + g_i - \sigma_i w_{t,i}$
    $n_i \leftarrow n_i + g_i^2$
  end for
end for

Per-Coordinate 意思是FTRL是对w每一维分开训练更新的，每一维使用的是不同的学习速率，也是上面代码中lamda2之前的那一项。与w所有特征维度使用统一的学习速率相比，这种方法考虑了训练样本本身在不同特征上分布的不均匀性，如果包含w某一个维度特征的训练样本很少，每一个样本都很珍贵，那么该特征维度对应的训练速率可以独自保持比较大的值，每来一个包含该特征的样本，就可以在该样本的梯度上前进一大步，而不需要与其他特征维度的前进步调强行保持一致。

1.5.3 简要理解

我们再看看下一时刻的特征权重的更新公式，增加理解（我个人觉得找到的这个解释相对易于理解）：

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}} \left(\mathbf{g}_{1:t} \cdot \mathbf{w} + \frac{1}{2} \sum_{s=1}^{t} \sigma_s \|\mathbf{w} - \mathbf{w}_s\|_2^2 + \lambda_1 \|\mathbf{w}\|_1 \right)$$

式中**第一项**是对损失函数的贡献的一个估计，**第二项**是控制w（也就是model）在每次迭代中变化不要太大，**第三项**代表L1正则（获得稀疏解）。

0x02 示例代码

我们采用的就是Alink官方示例代码。我们可以看到大致分为几部分：

- 建立特征处理管道，其包括StandardScaler和FeatureHasher，进行标准化缩放和特征哈希，最后得到了特征向量，这部分可以参见 Alink漫谈(九)： 特征工程 之 特征哈希/标准化缩放；
- 对于流数据源进行实时切分得到原始训练数据和原始预测数据，这部分可以参见 Alink漫谈(七)：如何划分训练数据集和测试数据集；
- 训练出一个逻辑回归模型作为FTRL算法的初始模型，这是为了系统冷启动的需要。逻辑回归和线性回归类似，所以大家可以参见这两篇文章： Alink漫谈(十)： 线性回归实现 之 数据预处理 和 Alink漫谈(十一)： 线性回归 之 L-BFGS优化
- 在初始模型基础上进行FTRL在线训练；
- 在FTRL在线模型的基础上，连接预测数据进行预测；
- 对预测结果流进行评估，具体可以参见 Alink漫谈(八)：二分类评估 AUC、K-S、PRC、Precision、Recall、LiftChart 如何实现；

你大概已经看出来了，为了剖析FTRL，我前面做了很多工作.....

```
public class FTRLExample {

    public static void main(String[] args) throws Exception {
        ......
        // setup feature engineering pipeline
        Pipeline featurePipeline = new Pipeline()
                .add(
                        new StandardScaler() // 标准缩放
                                .setSelectedCols(numericalColNames)
                )
                .add(
                        new FeatureHasher() // 特征哈希
                                .setSelectedCols(selectedColNames)
                                .setCategoricalCols(categoryColNames)
                                .setOutputCol(vecColName)
                                .setNumFeatures(numHashFeatures)
                );
        // 构建特征工程流水线
        // fit feature pipeline model
        PipelineModel featurePipelineModel = featurePipeline.fit(trainBatchData);

        // prepare stream train data
        CsvSourceStreamOp data = new CsvSourceStreamOp()
                .setFilePath("http://alink-release.oss-cn-beijing.aliyuncs.com/data-files/avazu
-ctr-train-8M.csv")
                .setSchemaStr(schemaStr)
                .setIgnoreFirstLine(true);

        // 对于流数据源进行实时切分得到原始训练数据和原始预测数据
        // split stream to train and eval data
        SplitStreamOp splitter = new SplitStreamOp().setFraction(0.5).linkFrom(data);

        // 训练出一个逻辑回归模型作为FTRL算法的初始模型，这是为了系统冷启动的需要。
        // train initial batch model
        LogisticRegressionTrainBatchOp lr = new LogisticRegressionTrainBatchOp()
                .setVectorCol(vecColName)
                .setLabelCol(labelColName)
                .setWithIntercept(true)
                .setMaxIter(10);
        BatchOperator<?> initModel = featurePipelineModel.transform(trainBatchData).link(lr);

        // 在初始模型基础上进行FTRL在线训练
        // ftrl train
        FtrlTrainStreamOp model = new FtrlTrainStreamOp(initModel)
                .setVectorCol(vecColName)
                .setLabelCol(labelColName)
                .setWithIntercept(true)
                .setAlpha(0.1)
                .setBeta(0.1)
                .setL1(0.01)
                .setL2(0.01)
```

```
                .setTimeInterval(10)
                .setVectorSize(numHashFeatures)
                .linkFrom(featurePipelineModel.transform(splitter));

        // 在FTRL在线模型的基础上，连接预测数据进行预测
        // ftrl predict
        FtrlPredictStreamOp predictResult = new FtrlPredictStreamOp(initModel)
                .setVectorCol(vecColName)
                .setPredictionCol("pred")
                .setReservedCols(new String[]{labelColName})
                .setPredictionDetailCol("details")
                .linkFrom(model, featurePipelineModel.transform(splitter.getSideOutput(0)));

        // 对预测结果流进行评估
        // ftrl eval
        predictResult
                .link(
                        new EvalBinaryClassStreamOp()
                                .setLabelCol(labelColName)
                                .setPredictionCol("pred")
                                .setPredictionDetailCol("details")
                                .setTimeInterval(10)
                )
                .link(
                        new JsonValueStreamOp()
                                .setSelectedCol("Data")
                                .setReservedCols(new String[]{"Statistics"})
                                .setOutputCols(new String[]{"Accuracy", "AUC", "ConfusionMatrix
"})
                                .setJsonPath(new String[]{"$.Accuracy", "$.AUC", "$.ConfusionMa
trix"})
                )
                .print();

        StreamOperator.execute();
    }
}
```

0x03 问题

用问题来引导剖析比较好，以下是我们容易想到的一些问题。

- 训练阶段和预测阶段都有预制模型以应对"冷启动"嘛?
- 训练阶段和预测阶段是如何关联起来的?
- 如何把训练出来的模型传给预测阶段?
- 输出模型时候，模型过大怎么处理?
- 在线训练的模型通过什么机制实现更新？是定时驱动更新嘛?
- 预测阶段加载模型过程中，还可以预测嘛？有没有机制保证这段时间内也能预测?
- 训练和预测中，有哪些阶段用到了并行处理?
- 遇到高维向量如何处理？切分开嘛?

后续我们会一一探究这些问题。

0x04 总体逻辑

在线训练是在 FtrlTrainStreamOp 类中实现的，其 linkFrom 函数实现了基本逻辑。

主要逻辑是：

- 1)加载初始化模型到 dataBridge；dataBridge = DirectReader.collect(model);
- 2)获取相关参数。比如vectorSize默认是30000，是否有hasInterceptItem；
- 3)获取切分信息。splitInfo = getSplitInfo(featureSize, hasInterceptItem, parallelism)；下面马上会用到。
- 4)切分高维向量。初始化数据做了特征哈希，会产生高维向量，这里需要进行切割。initData.flatMap(new SplitVector(splitInfo, hasInterceptItem, vectorSize,vectorTrainIdx, featureIdx, labelIdx))；
- 5)构建一个 IterativeStream.ConnectedIterativeStreams iteration，这样会构建（或者说连接）两个数据流：反馈流和训练流；
- 6)用iteration来构建迭代体 iterativeBody，其包括两部分：CalcTask，ReduceTask；
  - CalcTask分成两个部分。flatMap1 是分布计算FTRL迭代需要的predict，flatMap2 是FTRL的更新参数部分；
  - ReduceTask分为两个功能：归并这些predict计算结果 / 如果满足条件则归并模型并 & 向下游算子输出模型；
- 7)result = iterativeBody.filter；基本是以时间间隔为标准来判断（也可以认为是时间驱动），"时间未过期&向量有意义" 的数据将被发送回反馈数据流，继续迭代，**回到步骤 6)，进入flatMap2**；
- 8)output = iterativeBody.filter；符合标准（时间过期了）的数据将跳出迭代，然后算法会调用WriteModel将LineModelData转换为多条Row，转发给下游operator（也就是在线预测阶段）；**即定时把**

模型更新给在线预测阶段。

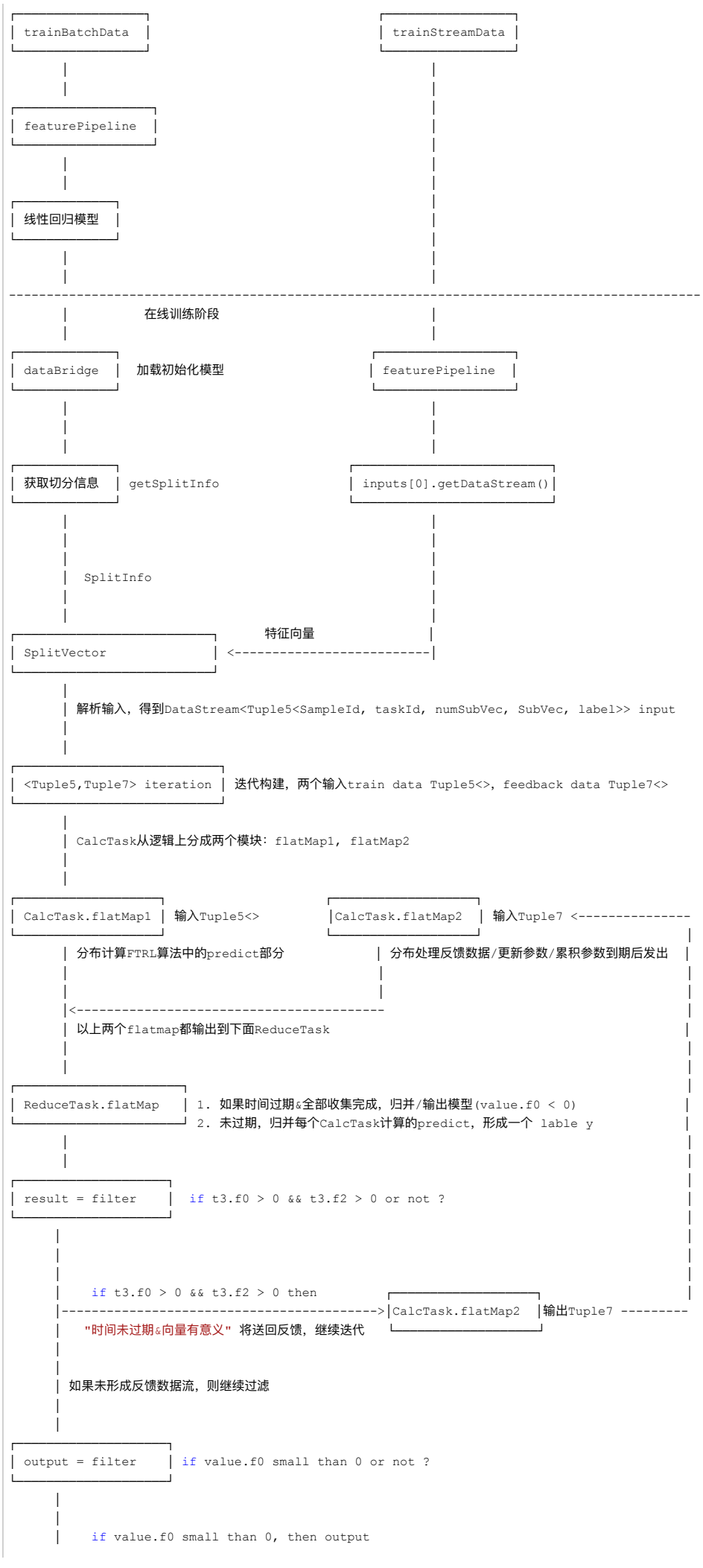
代码摘要是：

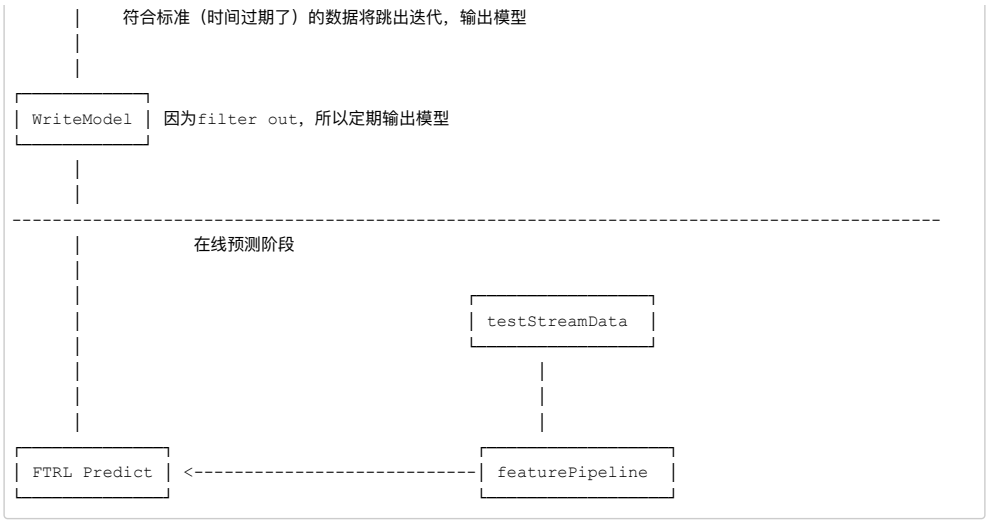
```
@Override
public FtrlTrainStreamOp linkFrom(StreamOperator<?>... inputs) {
    ......
    // 3)获取切分信息
    final int[] splitInfo = getSplitInfo(featureSize, hasInterceptItem, parallelism);

    DataStream<Row> initData = inputs[0].getDataStream();

    // 4)切分高维向量。
    // Tuple5<SampleId, taskId, numSubVec, SubVec, label>
    DataStream<Tuple5<Long, Integer, Integer, Vector, Object>> input
        = initData.flatMap(new SplitVector(splitInfo, hasInterceptItem, vectorSize,
        vectorTrainIdx, featureIdx, labelIdx))
        .partitionCustom(new CustomBlockPartitioner(), 1);

    // train data format = <sampleId, subSampleTaskId, subNum, SparseVector(subSample), label>
    // feedback format = Tuple7<sampleId, subSampleTaskId, subNum, SparseVector(subSample), lab
el, wx, timeStamps>
    // 5)构建一个 IterativeStream.ConnectedIterativeStreams iteration，这样会构建（或者说连接）两个数
据流：反馈流和训练流；
    IterativeStream.ConnectedIterativeStreams<Tuple5<Long, Integer, Integer, Vector, Object>,
        Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>>
        iteration = input.iterate(Long.MAX_VALUE)
        .withFeedbackType(TypeInformation
            .of(new TypeHint<Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>>() {}
));

    // 6)用iteration来构建迭代体 iterativeBody，其包括两部分：CalcTask, ReduceTask;
    DataStream iterativeBody = iteration.flatMap(
        new CalcTask(dataBridge, splitInfo, getParams()))
        .keyBy(0)
        .flatMap(new ReduceTask(parallelism, splitInfo))
        .partitionCustom(new CustomBlockPartitioner(), 1);

    // 7)result = iterativeBody.filter；基本是以时间间隔为标准来判断（也可以认为是时间驱动），"时间未过期&
向量有意义" 的数据将被发送回反馈数据流，继续迭代，回到步骤 6)，进入flatMap2;
    DataStream<Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>>
        result = iterativeBody.filter(
        new FilterFunction<Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>>() {
            @Override
            public boolean filter(Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>
t3)
                throws Exception {
                // if t3.f0 > 0 && t3.f2 > 0 then feedback
                return (t3.f0 > 0 && t3.f2 > 0);
            }
        });

    // 8)output = iterativeBody.filter；符合标准（时间过期了）的数据将跳出迭代，然后算法会调用WriteModel
将LineModelData转换为多条Row，转发给下游operator（也就是在线预测阶段）；即定时把模型更新给在线预测阶段。
    DataStream<Row> output = iterativeBody.filter(
        new FilterFunction<Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>>() {
            @Override
            public boolean filter(Tuple7<Long, Integer, Integer, Vector, Object, Double, Long>
value)
                throws Exception {
                /* if value.f0 small than 0, then output */
                return value.f0 < 0;
            }
        }).flatMap(new WriteModel(labelType, getVectorCol(), featureCols, hasInterceptItem));

    // 指定了某个流将成为迭代程序的结束，并且这个流将作为输入的第二部分（second input）被反馈回迭代
    iteration.closeWith(result);

    TableSchema schema = new LinearModelDataConverter(labelType).getModelSchema();

    ......
    this.setOutput(output, names, types);
    return this;
}
```

为了方便阅读，我们给出流程图如下（这里省略了split 训练数据集/测试数据集）：

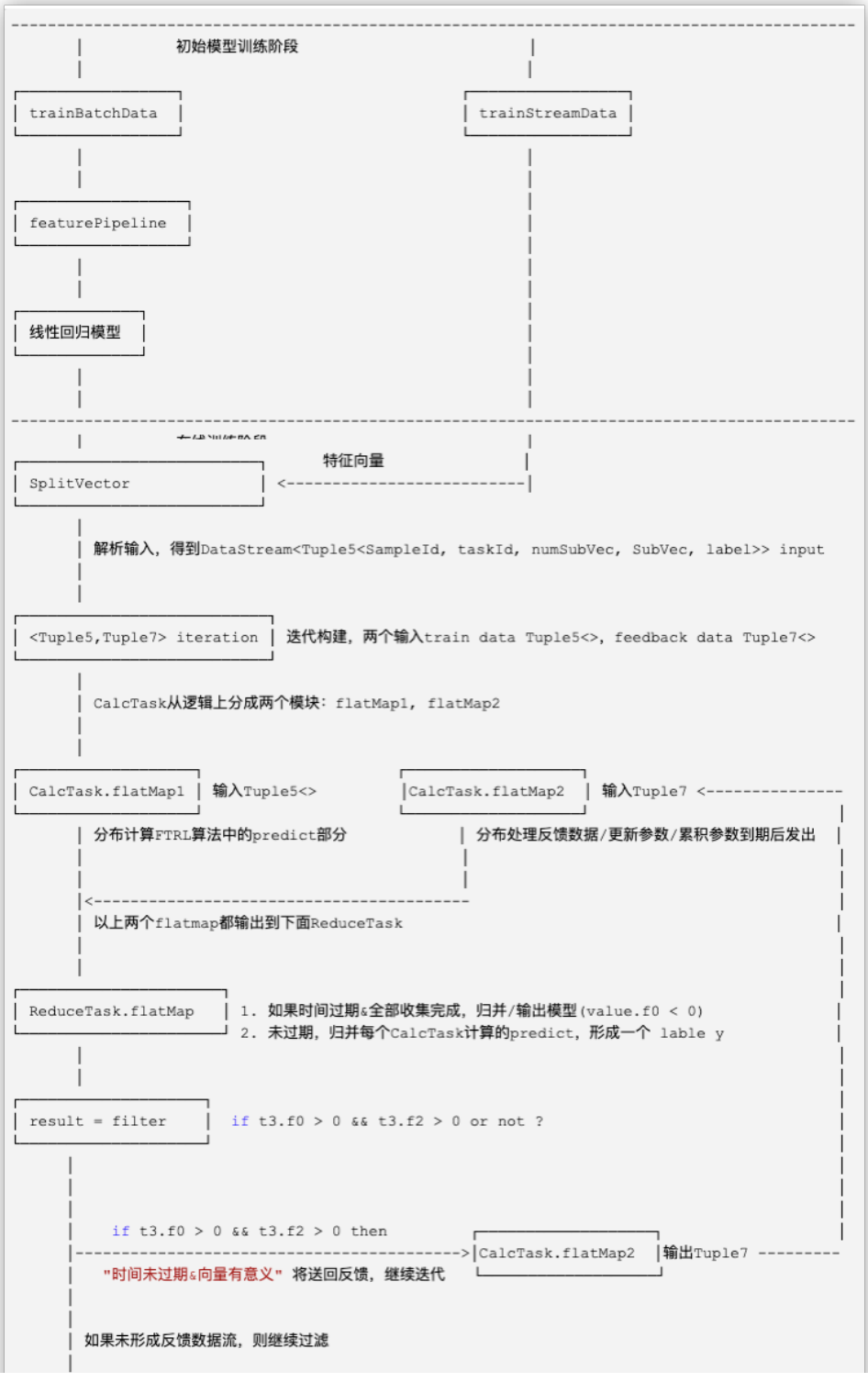
原谅我用这种办法画图，因为我最讨厌看到一篇好文，结果发现图没了...

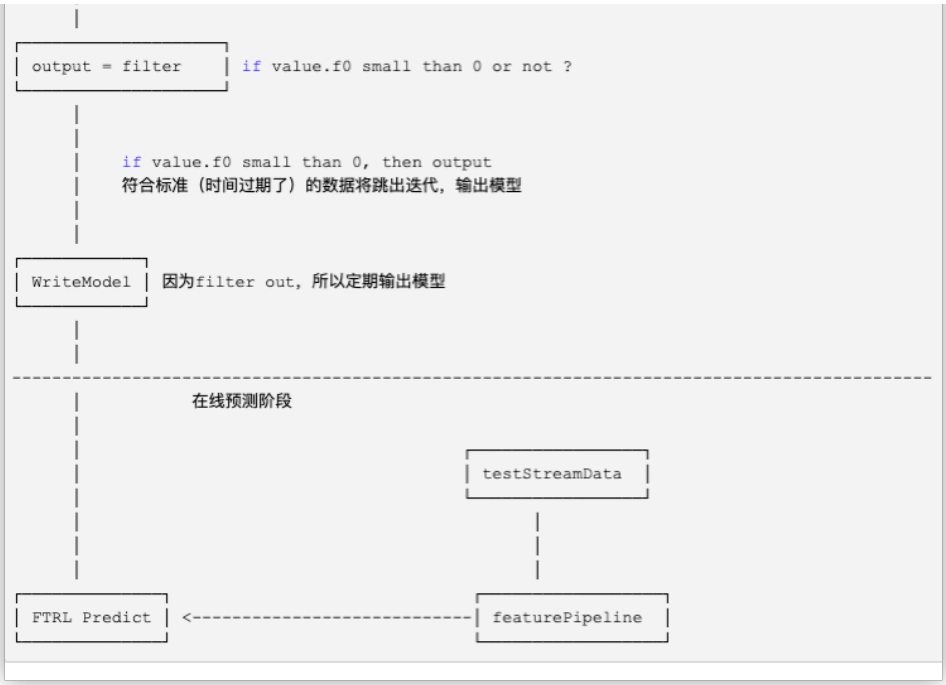
```
-------------------------------------------------------------------------------------------
       |            初始模型训练阶段                                  |
       |                                                            |
```


trainBatchData
trainStreamData
featurePipeline
线性回归模型
在线训练阶段
dataBridge
加载初始化模型
featurePipeline
获取切分信息
getSplitInfo
inputs[0].getDataStream()
SplitInfo
特征向量
SplitVector
解析输入，得到DataStream<Tuple5<SampleId, taskId, numSubVec, SubVec, label>> input
<Tuple5,Tuple7> iteration
迭代构建，两个输入train data Tuple5<>, feedback data Tuple7<>
CalcTask从逻辑上分成两个模块：flatMap1, flatMap2
CalcTask.flatMap1
输入Tuple5<>
CalcTask.flatMap2
输入Tuple7
分布计算FTRL算法中的predict部分
分布处理反馈数据/更新参数/累积参数到期后发出
以上两个flatmap都输出到下面ReduceTask
ReduceTask.flatMap
1. 如果时间过期&全部收集完成，归并/输出模型(value.f0 < 0)
2. 未过期，归并每个CalcTask计算的predict，形成一个 lable y
result = filter
if t3.f0 > 0 && t3.f2 > 0 or not ?
if t3.f0 > 0 && t3.f2 > 0 then
"时间未过期&向量有意义" 将送回反馈，继续迭代
CalcTask.flatMap2
输出Tuple7
如果未形成反馈数据流，则继续过滤
output = filter
if value.f0 small than 0 or not ?
if value.f0 small than 0, then output



因为上图在手机上会变形，以下这个图为大家在手机上浏览。





0xFF 参考

【机器学习】逻辑回归（非常详细）

逻辑回归(logistics regression)

【机器学习】LR的分布式（并行化）实现

并行逻辑回归

机器学习算法及其并行化讨论

Online LR—— FTRL 算法理解

在线优化算法 FTRL 的原理与实现

LR+FTRL算法原理以及工程化实现

Flink流处理之迭代API分析

FTRL公式推导

FTRL论文笔记

在线机器学习FTRL(Follow-the-regularized-Leader)算法介绍

FTRL代码实现

FTRL实战之LR+FTRL（代码采用的稠密数据）

在线学习算法FTRL-Proximal原理

基于FTRL的在线CTR预测算法

CTR预测算法之FTRL-Proximal

各大公司广泛使用的在线学习算法FTRL详解

在线最优化求解(Online Optimization)之五：FTRL

FOLLOW THE REGULARIZED LEADER (FTRL) 算法总结