Alink漫谈(八)：二分类评估 AUC、K-S、PRC、Precision、Recall、LiftChart 如何实现

目录

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。二分类评估是对二分类算法的预测结果进行效果评估。本文将剖析Alink中对应代码实现。

0x01 相关概念

如果对本文某些概念有疑惑，可以参见之前文章 [白话解析] 通过实例来梳理概念 ：准确率 (Accuracy)、精准率(Precision)、召回率(Recall) 和 F值(F-Measure)

0x02 示例代码

```java
public class EvalBinaryClassExample {

    AlgoOperator getData(boolean isBatch) {
        Row[] rows = new Row[]{
                Row.of("prefix1", "{\"prefix1\": 0.9, \"prefix0\": 0.1}"),
                Row.of("prefix1", "{\"prefix1\": 0.8, \"prefix0\": 0.2}"),
                Row.of("prefix1", "{\"prefix1\": 0.7, \"prefix0\": 0.3}"),
                Row.of("prefix0", "{\"prefix1\": 0.75, \"prefix0\": 0.25}"),
                Row.of("prefix0", "{\"prefix1\": 0.6, \"prefix0\": 0.4}")
        };

        String[] schema = new String[]{"label", "detailInput"};

        if (isBatch) {
            return new MemSourceBatchOp(rows, schema);
        } else {
            return new MemSourceStreamOp(rows, schema);
        }
    }

    public static void main(String[] args) throws Exception {
        EvalBinaryClassExample test = new EvalBinaryClassExample();
        BatchOperator batchData = (BatchOperator) test.getData(true);

        BinaryClassMetrics metrics = new EvalBinaryClassBatchOp()
                .setLabelCol("label")
                .setPredictionDetailCol("detailInput")
                .linkFrom(batchData)
                .collectMetrics();

        System.out.println("RocCurve:" + metrics.getRocCurve());
        System.out.println("AUC:" + metrics.getAuc());
        System.out.println("KS:" + metrics.getKs());
        System.out.println("PRC:" + metrics.getPrc());
        System.out.println("Accuracy:" + metrics.getAccuracy());
        System.out.println("Macro Precision:" + metrics.getMacroPrecision());
        System.out.println("Micro Recall:" + metrics.getMicroRecall());
        System.out.println("Weighted Sensitivity:" + metrics.getWeightedSensitivity());
    }
}
```

程序输出

```
RocCurve:([0.0, 0.0, 0.0, 0.5, 0.5, 1.0, 1.0],[0.0, 0.3333333333333333, 0.6666666666666666, 0.6666666666666666, 1.0, 1.0, 1.0])
AUC:0.8333333333333333
KS:0.6666666666666666
PRC:0.9027777777777777
Accuracy:0.6
Macro Precision:0.3
Micro Recall:0.6
Weighted Sensitivity:0.6
```

在 Alink 中，二分类评估有批处理，流处理两种实现，下面一一为大家介绍（ Alink 复杂之一在于大量精细的数据结构，所以下文会大量打印程序中变量以便大家理解）。

2.1 主要思路

- 把 [0,1] 分成假设 100000个桶(bin)。所以得到positiveBin / negativeBin 两个100000的数组。
- 根据输入给positiveBin / negativeBin赋值。positiveBin就是 TP + FP，negativeBin就是 TN + FN。这些是后续计算的基础。
- 遍历bins中每一个有意义的点，计算出totalTrue和totalFalse，并且在每一个点上计算该点的混淆矩阵，tpr，以及rocCurve，recallPrecisionCurve，liftChart在该点对应的数据；
- 依据曲线内容计算并且存储 AUC/PRC/KS

具体后续还有详细调用关系综述。

0x03 批处理

3.1 EvalBinaryClassBatchOp

EvalBinaryClassBatchOp是二分类评估的实现，功能是计算二分类的评估指标(evaluation metrics)。

输入有两种：

- label column and predResult column
- label column and predDetail column。如果有predDetail，则predResult被忽略

我们例子中 `"prefix1"` 就是 label， `"{\"prefix1\": 0.9, \"prefix0\": 0.1}"` 就是 predDetail

```
Row.of("prefix1", "{\"prefix1\": 0.9, \"prefix0\": 0.1}")
```

具体类摘录如下：

```
public class EvalBinaryClassBatchOp extends BaseEvalClassBatchOp<EvalBinaryClassBatchOp> implements BinaryEvaluationParams <EvalBinaryClassBatchOp>, EvaluationMetricsCollector<BinaryClassMetrics> {

	@Override
	public BinaryClassMetrics collectMetrics() {
		return new BinaryClassMetrics(this.collect().get(0));
	}
}
```

可以看到，其主要工作都是在基类BaseEvalClassBatchOp中完成，所以我们会首先看BaseEvalClassBatchOp。

3.2 BaseEvalClassBatchOp

我们还是从 linkFrom 函数入手，其主要是做了几件事：

- 获取配置信息
- 从输入中提取某些列："label"，"detailInput"
- calLabelPredDetailLocal会按照partition分别计算evaluation metrics
- 综合reduce上述计算结果
- SaveDataAsParams函数会把最终数值输入到 output table

具体代码如下

```
@Override
public T linkFrom(BatchOperator<?>... inputs) {
    BatchOperator<?> in = checkAndGetFirst(inputs);
    String labelColName = this.get(MultiEvaluationParams.LABEL_COL);
    String positiveValue = this.get(BinaryEvaluationParams.POS_LABEL_VAL_STR);

    // Judge the evaluation type from params.
    ClassificationEvaluationUtil.Type type = ClassificationEvaluationUtil.judgeEvaluationType(t
his.getParams());

    DataSet<BaseMetricsSummary> res;
    switch (type) {
        case PRED_DETAIL: {
            String predDetailColName = this.get(MultiEvaluationParams.PREDICTION_DETAIL_COL);
            // 从输入中提取某些列："label"，"detailInput"
            DataSet<Row> data = in.select(new String[] {labelColName, predDetailColName}).getDa
taSet();
            // 按照partition分别计算evaluation metrics
            res = calLabelPredDetailLocal(data, positiveValue, binary);
            break;
        }
        ......
    }

    // 综合reduce上述计算结果
    DataSet<BaseMetricsSummary> metrics = res
        .reduce(new EvaluationUtil.ReduceBaseMetrics());

    // 把最终数值输入到 output table
    this.setOutput(metrics.flatMap(new EvaluationUtil.SaveDataAsParams()),
        new String[] {DATA_OUTPUT}, new TypeInformation[] {Types.STRING});

    return (T)this;
}

// 执行中一些变量如下
labelColName = "label"
predDetailColName = "detailInput"
type = {ClassificationEvaluationUtil$Type@2532} "PRED_DETAIL"
binary = true
positiveValue = null
```

3.2.0 调用关系综述

<u>**因为后续代码调用关系复杂，所以先给出一个调用关系**</u>：

- 从输入中提取某些列："label"，"detailInput"，in.select(new String[] {labelColName, predDetailColName}).getDataSet()。因为可能输入还有其他列，而只有某些列是我们计算需要的，所以只提取这些列。
- 按照partition分别计算evaluation metrics，即调用 calLabelPredDetailLocal(data, positiveValue, binary);
    - flatMap会从label列和prediction列中，取出所有labels（注意是取出labels的名字），发送给下游算子。
    - reduceGroup主要功能是通过 buildLabelIndexLabelArray 去重 "labels名字"，然后给每一个label一个ID，得到一个 <labels, ID>的map，最后返回是二元组(map, labels)，即({prefix1=0, prefix0=1},[prefix1, prefix0])。从后文看，<labels, ID>Map看来是多分类才用到。二分类只用到了labels。
    - mapPartition 分区调用 CalLabelDetailLocal 来计算混淆矩阵，主要是分区调用

getDetailStatistics，前文中得到的二元组(map, labels)会作为参数传递进来 。

- getDetailStatistics 遍历 rows 数据，提取每一个item（比如 "prefix1,{"prefix1": 0.8, "prefix0": 0.2}"），然后通过updateBinaryMetricsSummary累积计算混淆矩阵所需数据。
  - updateBinaryMetricsSummary 把 [0,1] 分成假设 100000个桶(bin)。所以得到 positiveBin / negativeBin 两个100000的数组。positiveBin就是 TP + FP，negativeBin 就是 TN + FN。
    - 如果某个 sample 为 正例 (positive value) 的概率是 p, 则该 sample 对应的 bin index 就是 p * 100000。如果 p 被预测为正例 (positive value)，则 positiveBin[index]++,
    - 否则就是被预测为负例(negative value)，则negativeBin[index]++。

- 综合reduce上述计算结果，metrics = res.reduce(new EvaluationUtil.ReduceBaseMetrics());
  - 具体计算是在BinaryMetricsSummary.merge，其作用就是Merge the bins, and add the logLoss。
- 把最终数值输入到 output table，setOutput(metrics.flatMap(new EvaluationUtil.SaveDataAsParams()..);
  - 归并所有BaseMetrics后，得到total BaseMetrics，计算indexes存入params。collector.collect(t.toMetrics().serialize());
    - 实际业务在BinaryMetricsSummary.toMetrics，即基于bin的信息计算，然后存储到params。
      - extractMatrixThreCurve函数取出非空的bins，据此计算出ConfusionMatrix array（混淆矩阵），threshold array, rocCurve/recallPrecisionCurve/LiftChart.
        - 遍历bins中每一个有意义的点，计算出totalTrue和totalFalse，并且在每一个点上计算：
        - curTrue += positiveBin[index]; curFalse += negativeBin[index];
        - 得到该点的混淆矩阵 new ConfusionMatrix(new long[][] {{curTrue, curFalse}, {totalTrue - curTrue, totalFalse - curFalse}});
        - 得到 tpr = (totalTrue == 0 ? 1.0 : 1.0 * curTrue / totalTrue);
        - rocCurve，recallPrecisionCurve，liftChart在该点对应的数据；
      - 依据曲线内容计算并且存储 AUC/PRC/KS
      - 对生成的rocCurve/recallPrecisionCurve/LiftChart输出进行抽样
      - 依据抽样后的输出存储 RocCurve/RecallPrecisionCurve/LiftChar
      - 存储正例样本的度量指标
      - 存储Logloss
      - Pick the middle point where threshold is 0.5.

3.2.1 calLabelPredDetailLocal

本函数按照partition分别计算评估指标 evaluation metrics。是的，这代码很短，但是有个地方需要注意。有时候越简单的地方越容易疏漏。容易疏漏点是：

第一行代码的结果 labels 是第二行代码的参数，而并非第二行主体。第二行代码主体和第一行代码主体一样，都是data。

```
private static DataSet<BaseMetricsSummary> calLabelPredDetailLocal(DataSet<Row> data, final String positiveValue, oolean binary) {

    DataSet<Tuple2<Map<String, Integer>, String[]>> labels = data.flatMap(new FlatMapFunction<Row, String>() {
        @Override
        public void flatMap(Row row, Collector<String> collector) {
            TreeMap<String, Double> labelProbMap;
            if (EvaluationUtil.checkRowFieldNotNull(row)) {
                labelProbMap = EvaluationUtil.extractLabelProbMap(row);
                labelProbMap.keySet().forEach(collector::collect);
                collector.collect(row.getField(0).toString());
```

```
            }
        }
    }).reduceGroup(new EvaluationUtil.DistinctLabelIndexMap(binary, positiveValue));

    return data
        .rebalance()
        .mapPartition(new CalLabelDetailLocal(binary))
        .withBroadcastSet(labels, LABELS);
}
```

calLabelPredDetailLocal中具体分为三步骤:

- 在flatMap会从label列和prediction列中，取出所有labels（注意是取出labels的名字 ），发送给下游算子。
- reduceGroup的主要功能是去重 "labels名字"，然后给每一个label一个ID，最后结果是一个<labels, ID>Map。
- mapPartition 是分区调用 CalLabelDetailLocal 来计算混淆矩阵。

下面具体看看。

3.2.1.1 flatMap

在flatMap中，主要是从label列和prediction列中，取出所有labels（注意是取出labels的名字 ），发送给下游算子。

EvaluationUtil.extractLabelProbMap 作用就是解析输入的json，获得具体detailInput中的信息。

下游算子是reduceGroup，所以Flink runtime会对这些labels自动去重。如果对这部分有兴趣，可以参见我之前介绍reduce的文章。CSDN： [源码解析] Flink的groupBy和reduce究竟做了什么 博客园：[源码解析] Flink的groupBy和reduce究竟做了什么

程序中变量如下

```
row = {Row@8922} "prefix1,{"prefix1": 0.9, "prefix0": 0.1}"
 fields = {Object[2]@8925}
  0 = "prefix1"
  1 = "{"prefix1": 0.9, "prefix0": 0.1}"

labelProbMap = {TreeMap@9008}  size = 2
 "prefix0" -> {Double@9015} 0.1
 "prefix1" -> {Double@9017} 0.9

labelProbMap.keySet().forEach(collector::collect); //这里发送 "prefix0", "prefix1"
collector.collect(row.getField(0).toString());  // 这里发送 "prefix1"
// 因为下一个操作是reduceGroup，所以这些label会被runtime去重
```

3.2.1.2 reduceGroup

主要功能是通过buildLabelIndexLabelArray去重labels，然后给每一个label一个ID，最后结果是一个<labels, ID>的Map。

```
reduceGroup(new EvaluationUtil.DistinctLabelIndexMap(binary, positiveValue));
```

DistinctLabelIndexMap的作用是从label列和prediction列中，取出所有不同的labels，返回一个<labels, ID>的map，根据后续代码看，这个map是多分类才用到。Get all the distinct labels from label column and prediction column, and return the map of labels and their IDs.

前面已经提到，这里的参数rows已经被自动去重。

```
public static class DistinctLabelIndexMap implements
    GroupReduceFunction<String, Tuple2<Map<String, Integer>, String[]>> {
    ......
    @Override
    public void reduce(Iterable<String> rows, Collector<Tuple2<Map<String, Integer>, String[]>>
 collector) throws Exception {
        HashSet<String> labels = new HashSet<>();
        rows.forEach(labels::add);
        collector.collect(buildLabelIndexLabelArray(labels, binary, positiveValue));
    }
}

// 变量为
labels = {HashSet@9008}  size = 2
 0 = "prefix1"
 1 = "prefix0"
binary = true
```

buildLabelIndexLabelArray的作用是给每一个label一个ID，得到一个 <labels, ID>的map，最后返回是二元组(map, labels)，即({prefix1=0, prefix0=1},[prefix1, prefix0])。

```
// Give each label an ID, return a map of label and ID.
public static Tuple2<Map<String, Integer>, String[]> buildLabelIndexLabelArray(HashSet<String>
set,boolean binary, String positiveValue) {
    String[] labels = set.toArray(new String[0]);
    Arrays.sort(labels, Collections.reverseOrder());

    Map<String, Integer> map = new HashMap<>(labels.length);
    if (binary && null != positiveValue) {
        if (labels[1].equals(positiveValue)) {
            labels[1] = labels[0];
            labels[0] = positiveValue;
        }
        map.put(labels[0], 0);
        map.put(labels[1], 1);
    } else {
        for (int i = 0; i < labels.length; i++) {
            map.put(labels[i], i);
        }
    }
    return Tuple2.of(map, labels);
}

// 程序变量如下
labels = {String[2]@9013}
 0 = "prefix1"
 1 = "prefix0"
map = {HashMap@9014}  size = 2
 "prefix1" -> {Integer@9020} 0
 "prefix0" -> {Integer@9021} 1
```

3.2.1.3 mapPartition

这里主要功能是分区调用 CalLabelDetailLocal 来为后来计算混淆矩阵做准备。

```
return data
    .rebalance()
    .mapPartition(new CalLabelDetailLocal(binary)) //这里是业务所在
    .withBroadcastSet(labels, LABELS);
```

具体工作是 CalLabelDetailLocal 完成的，其作用是分区调用getDetailStatistics

```
// Calculate the confusion matrix based on the label and predResult.
static class CalLabelDetailLocal extends RichMapPartitionFunction<Row, BaseMetricsSummary> {
        private Tuple2<Map<String, Integer>, String[]> map;
        private boolean binary;

        @Override
        public void open(Configuration parameters) throws Exception {
            List<Tuple2<Map<String, Integer>, String[]>> list = getRuntimeContext().getBroadcas
tVariable(LABELS);
            this.map = list.get(0);// 前文生成的二元组(map, labels)
        }

        @Override
        public void mapPartition(Iterable<Row> rows, Collector<BaseMetricsSummary> collector) {
            // 调用到了 getDetailStatistics
            collector.collect(getDetailStatistics(rows, binary, map));
        }
    }
```

getDetailStatistics 的作用是：初始化分类评估的度量指标 base classification evaluation metrics，累积计算混淆矩阵需要的数据。主要就是遍历 rows 数据，提取每一个item（比如 "prefix1,{"prefix1": 0.8, "prefix0": 0.2}"），然后累积计算混淆矩阵所需数据。

```
// Initialize the base classification evaluation metrics. There are two cases: BinaryClassMetri
cs and MultiClassMetrics.
    private static BaseMetricsSummary getDetailStatistics(Iterable<Row> rows,
                                                          String positiveValue,
                                                          boolean binary,
                                                          Tuple2<Map<String, Integer>, String[]> tuple) {
        BinaryMetricsSummary binaryMetricsSummary = null;
        MultiMetricsSummary multiMetricsSummary = null;
        Tuple2<Map<String, Integer>, String[]> labelIndexLabelArray = tuple;  // 前文生成的二元组(
map, labels)

        Iterator<Row> iterator = rows.iterator();
        Row row = null;
        while (iterator.hasNext() && !checkRowFieldNotNull(row)) {
            row = iterator.next();
        }

        Map<String, Integer> labelIndexMap = null;
        if (binary) {
           // 二分法在这里
            binaryMetricsSummary = new BinaryMetricsSummary(
                new long[ClassificationEvaluationUtil.DETAIL_BIN_NUMBER],
                new long[ClassificationEvaluationUtil.DETAIL_BIN_NUMBER],
                labelIndexLabelArray.f1, 0.0, 0L);
        } else {
            //
            labelIndexMap = labelIndexLabelArray.f0; // 前文生成的<labels, ID>Map看来是多分类才用到
。
            multiMetricsSummary = new MultiMetricsSummary(
                new long[labelIndexMap.size()][labelIndexMap.size()],
                labelIndexLabelArray.f1, 0.0, 0L);
        }

        while (null != row) {
```

```
            if (checkRowFieldNotNull(row)) {
                TreeMap<String, Double> labelProbMap = extractLabelProbMap(row);
                String label = row.getField(0).toString();
                if (ArrayUtils.indexOf(labelIndexLabelArray.f1, label) >= 0) {
                    if (binary) {
                        // 二分法在这里
                        updateBinaryMetricsSummary(labelProbMap, label, binaryMetricsSummary);
                    } else {
                        updateMultiMetricsSummary(labelProbMap, label, labelIndexMap, multiMetr
icsSummary);
                    }
                }
            }
            row = iterator.hasNext() ? iterator.next() : null;
        }

        return binary ? binaryMetricsSummary : multiMetricsSummary;
}

//变量如下
tuple = {Tuple2@9252} "({prefix1=0, prefix0=1},[prefix1, prefix0])"
 f0 = {HashMap@9257}  size = 2
  "prefix1" -> {Integer@9264} 0
  "prefix0" -> {Integer@9266} 1
 f1 = {String[2]@9258}
  0 = "prefix1"
  1 = "prefix0"

row = {Row@9271} "prefix1,{"prefix1": 0.8, "prefix0": 0.2}"
 fields = {Object[2]@9276}
  0 = "prefix1"
  1 = "{"prefix1": 0.8, "prefix0": 0.2}"

labelIndexLabelArray = {Tuple2@9240} "({prefix1=0, prefix0=1},[prefix1, prefix0])"
 f0 = {HashMap@9288}  size = 2
  "prefix1" -> {Integer@9294} 0
  "prefix0" -> {Integer@9296} 1
 f1 = {String[2]@9242}
  0 = "prefix1"
  1 = "prefix0"

labelProbMap = {TreeMap@9342}  size = 2
 "prefix0" -> {Double@9378} 0.1
 "prefix1" -> {Double@9380} 0.9
```

先回忆下混淆矩阵：

			预测值 0	**预测值 1**
		真实值 0	TN	FP
		真实值 1	FN	TP

针对混淆矩阵，BinaryMetricsSummary 的作用是Save the evaluation data for binary classification。函数具体计算思路是：

- 把 [0,1] 分成ClassificationEvaluationUtil.DETAIL_BIN_NUMBER（100000）这么多桶(bin)。所以binaryMetricsSummary的positiveBin/negativeBin分别是两个100000的数组。如果某一个 sample 为

正例(positive value) 的概率是 p, 则该 sample 对应的 bin index 就是 p * 100000。如果 p 被预测为正例(positive value)，则positiveBin[index]++，否则就是被预测为负例(negative value)，则negativeBin[index]++。positiveBin就是 TP + FP，negativeBin就是 TN + FN。

- 所以这里会遍历输入，如果某一个输入（以 `"prefix1", "{\"prefix1\": 0.9, \"prefix0\": 0.1}"` 为例），0.9 是prefix1(正例) 的概率，0.1 是为prefix0(负例) 的概率。
  - 既然这个算法选择了 prefix1(正例)，所以就说明此算法是判别成 positive 的，所以在 positiveBin 的 90000 处 + 1。
  - 假设这个算法选择了 prefix0(负例)，则说明此算法是判别成 negative 的，所以应该在 negativeBin 的 90000 处 + 1。

具体对应我们示例代码的5个采样，分类如下：

```
Row.of("prefix1", "{\"prefix1\": 0.9, \"prefix0\": 0.1}"),  positiveBin 90000处+1
Row.of("prefix1", "{\"prefix1\": 0.8, \"prefix0\": 0.2}"),  positiveBin 80000处+1
Row.of("prefix1", "{\"prefix1\": 0.7, \"prefix0\": 0.3}"),  positiveBin 70000处+1
Row.of("prefix0", "{\"prefix1\": 0.75, \"prefix0\": 0.25}"), negativeBin 75000处+1
Row.of("prefix0", "{\"prefix1\": 0.6, \"prefix0\": 0.4}")  negativeBin 60000处+1
```

具体代码如下

```
public static void updateBinaryMetricsSummary(TreeMap<String, Double> labelProbMap,
                                              String label,
                                              BinaryMetricsSummary binaryMetricsSummary) {
    binaryMetricsSummary.total++;
    binaryMetricsSummary.logLoss += extractLogloss(labelProbMap, label);

    double d = labelProbMap.get(binaryMetricsSummary.labels[0]);
    int idx = d == 1.0 ? ClassificationEvaluationUtil.DETAIL_BIN_NUMBER - 1 :
        (int)Math.floor(d * ClassificationEvaluationUtil.DETAIL_BIN_NUMBER);
    if (idx >= 0 && idx < ClassificationEvaluationUtil.DETAIL_BIN_NUMBER) {
        if (label.equals(binaryMetricsSummary.labels[0])) {
            binaryMetricsSummary.positiveBin[idx] += 1;
        } else if (label.equals(binaryMetricsSummary.labels[1])) {
            binaryMetricsSummary.negativeBin[idx] += 1;
        } else {
                                            .....
        }
    }
}

private static double extractLogloss(TreeMap<String, Double> labelProbMap, String label) {
   Double prob = labelProbMap.get(label);
   prob = null == prob ? 0. : prob;
   return -Math.log(Math.max(Math.min(prob, 1 - LOG_LOSS_EPS), LOG_LOSS_EPS));
}

// 变量如下
ClassificationEvaluationUtil.DETAIL_BIN_NUMBER=100000

// 当 "prefix1", "{\"prefix1\": 0.9, \"prefix0\": 0.1}" 时候
labelProbMap = {TreeMap@9305}  size = 2
 "prefix0" -> {Double@9331} 0.1
 "prefix1" -> {Double@9333} 0.9

d = 0.9
idx = 90000
binaryMetricsSummary = {BinaryMetricsSummary@9262}
```

```
 labels = {String[2]@9242}
  0 = "prefix1"
  1 = "prefix0"
 total = 1
 positiveBin = {long[100000]@9263}  // 90000处+1
 negativeBin = {long[100000]@9264}
 logLoss = 0.10536051565782628

// 当 "prefix0", "{\"prefix1\": 0.6, \"prefix0\": 0.4}" 时候
labelProbMap = {TreeMap@9514}  size = 2
 "prefix0" -> {Double@9546} 0.4
 "prefix1" -> {Double@9547} 0.6

d = 0.6
idx = 60000
 binaryMetricsSummary = {BinaryMetricsSummary@9262}
 labels = {String[2]@9242}
  0 = "prefix1"
  1 = "prefix0"
 total = 2
 positiveBin = {long[100000]@9263}
 negativeBin = {long[100000]@9264} // 60000处+1
 logLoss = 1.0216512475319812
```

3.2.2 ReduceBaseMetrics

ReduceBaseMetrics作用是把局部计算的 BaseMetrics 聚合起来。

```
DataSet<BaseMetricsSummary> metrics = res
    .reduce(new EvaluationUtil.ReduceBaseMetrics());
```

ReduceBaseMetrics如下

```
public static class ReduceBaseMetrics implements ReduceFunction<BaseMetricsSummary> {
    @Override
    public BaseMetricsSummary reduce(BaseMetricsSummary t1, BaseMetricsSummary t2) throws Excep
tion {
        return null == t1 ? t2 : t1.merge(t2);
    }
}
```

具体计算是在BinaryMetricsSummary.merge，其作用就是Merge the bins, and add the logLoss。

```
@Override
public BinaryMetricsSummary merge(BinaryMetricsSummary binaryClassMetrics) {
    for (int i = 0; i < this.positiveBin.length; i++) {
        this.positiveBin[i] += binaryClassMetrics.positiveBin[i];
    }
    for (int i = 0; i < this.negativeBin.length; i++) {
        this.negativeBin[i] += binaryClassMetrics.negativeBin[i];
    }
    this.logLoss += binaryClassMetrics.logLoss;
    this.total += binaryClassMetrics.total;
    return this;
}

// 程序变量是
this = {BinaryMetricsSummary@9316}
 labels = {String[2]@9322}
  0 = "prefix1"
```

```
 1 = "prefix0"
total = 2
positiveBin = {long[100000]@9320}
negativeBin = {long[100000]@9323}
logLoss = 1.742969305058623
```

3.2.3 SaveDataAsParams

```
this.setOutput(metrics.flatMap(new EvaluationUtil.SaveDataAsParams()),
    new String[] {DATA_OUTPUT}, new TypeInformation[] {Types.STRING});
```

当归并所有BaseMetrics之后，得到了total BaseMetrics，计算indexes，存入到params。

```
public static class SaveDataAsParams implements FlatMapFunction<BaseMetricsSummary, Row> {
    @Override
    public void flatMap(BaseMetricsSummary t, Collector<Row> collector) throws Exception {
        collector.collect(t.toMetrics().serialize());
    }
}
```

实际业务在BinaryMetricsSummary.toMetrics中完成，即基于bin的信息计算，得到confusionMatrix array, threshold array, rocCurve/recallPrecisionCurve/LiftChart等等，然后存储到params。

```
public BinaryClassMetrics toMetrics() {
    Params params = new Params();
    // 生成若干曲线，比如rocCurve/recallPrecisionCurve/LiftChart
    Tuple3<ConfusionMatrix[], double[], EvaluationCurve[]> matrixThreCurve =
        extractMatrixThreCurve(positiveBin, negativeBin, total);

    // 依据曲线内容计算并且存储 AUC/PRC/KS
    setCurveAreaParams(params, matrixThreCurve.f2);

    // 对生成的rocCurve/recallPrecisionCurve/LiftChart输出进行抽样
    Tuple3<ConfusionMatrix[], double[], EvaluationCurve[]> sampledMatrixThreCurve = sample(
        PROBABILITY_INTERVAL, matrixThreCurve);

    // 依据抽样后的输出存储 RocCurve/RecallPrecisionCurve/LiftChar
    setCurvePointsParams(params, sampledMatrixThreCurve);
    ConfusionMatrix[] matrices = sampledMatrixThreCurve.f0;

    // 存储正例样本的度量指标
    setComputationsArrayParams(params, sampledMatrixThreCurve.f1, sampledMatrixThreCurve.f0);

    // 存储Logloss
    setLoglossParams(params, logLoss, total);

    // Pick the middle point where threshold is 0.5.
    int middleIndex = getMiddleThresholdIndex(sampledMatrixThreCurve.f1);
    setMiddleThreParams(params, matrices[middleIndex], labels);
    return new BinaryClassMetrics(params);
}
```

extractMatrixThreCurve是全文重点。这里是 Extract the bins who are not empty, keep the middle threshold 0.5，然后初始化了 RocCurve, Recall-Precision Curve and Lift Curve，计算出 ConfusionMatrix array（混淆矩阵），threshold array, rocCurve/recallPrecisionCurve/LiftChart.。

```
/**
 * Extract the bins who are not empty, keep the middle threshold 0.5.
 * Initialize the RocCurve, Recall-Precision Curve and Lift Curve.
```

```
 * RocCurve: (FPR, TPR), starts with (0,0). Recall-Precision Curve: (recall, precision), starts
 with (0, p), p is the precision with the lowest. LiftChart: (TP+FP/total, TP), starts with (0,
0). confusion matrix = [TP FP][FN * TN].
 *
 * @param positiveBin positiveBins.
 * @param negativeBin negativeBins.
 * @param total       sample number
 * @return ConfusionMatrix array, threshold array, rocCurve/recallPrecisionCurve/LiftChart.
 */
static Tuple3<ConfusionMatrix[], double[], EvaluationCurve[]> extractMatrixThreCurve(long[] pos
itiveBin, long[] negativeBin, long total) {
    ArrayList<Integer> effectiveIndices = new ArrayList<>();
    long totalTrue = 0, totalFalse = 0;

    // 计算totalTrue, totalFalse, effectiveIndices
    for (int i = 0; i < ClassificationEvaluationUtil.DETAIL_BIN_NUMBER; i++) {
        if (0L != positiveBin[i] || 0L != negativeBin[i]
            || i == ClassificationEvaluationUtil.DETAIL_BIN_NUMBER / 2) {
            effectiveIndices.add(i);
            totalTrue += positiveBin[i];
            totalFalse += negativeBin[i];
        }
    }

// 以我们例子，得到
effectiveIndices = {ArrayList@9273}  size = 6
 0 = {Integer@9277} 50000 //这里加入了中间点
 1 = {Integer@9278} 60000
 2 = {Integer@9279} 70000
 3 = {Integer@9280} 75000
 4 = {Integer@9281} 80000
 5 = {Integer@9282} 90000
totalTrue = 3
totalFalse = 2

    // 继续初始化，生成若干curve
    final int length = effectiveIndices.size();
    final int newLen = length + 1;
    final double m = 1.0 / ClassificationEvaluationUtil.DETAIL_BIN_NUMBER;
    EvaluationCurvePoint[] rocCurve = new EvaluationCurvePoint[newLen];
    EvaluationCurvePoint[] recallPrecisionCurve = new EvaluationCurvePoint[newLen];
    EvaluationCurvePoint[] liftChart = new EvaluationCurvePoint[newLen];
    ConfusionMatrix[] data = new ConfusionMatrix[newLen];
    double[] threshold = new double[newLen];
    long curTrue = 0;
    long curFalse = 0;

// 以我们例子，得到
length = 6
newLen = 7
m = 1.0E-5

    // 计算，其中rocCurve, recallPrecisionCurve, liftChart 都可以从代码中看出
    for (int i = 1; i < newLen; i++) {
        int index = effectiveIndices.get(length - i);
        curTrue += positiveBin[index];
        curFalse += negativeBin[index];
        threshold[i] = index * m;
        // 计算出混淆矩阵
        data[i] = new ConfusionMatrix(
```

```
            new long[][] {{curTrue, curFalse}, {totalTrue - curTrue, totalFalse - curFalse}});
        double tpr = (totalTrue == 0 ? 1.0 : 1.0 * curTrue / totalTrue);
        // 比如当 90000 这点，得到 curTrue = 1 curFalse = 0 i = 1 index = 90000 tpr = 0.3333333333
333333。totalTrue = 3 totalFalse = 2,
        // 我们也知道，TPR = TP / (TP + FN) ，所以可以计算 tpr = 1 / 3
        rocCurve[i] = new EvaluationCurvePoint(totalFalse == 0 ? 1.0 : 1.0 * curFalse / totalFa
lse, tpr, threshold[i]);
        recallPrecisionCurve[i] = new EvaluationCurvePoint(tpr, curTrue + curTrue == 0 ? 1.0 :
1.0 * curTrue / (curTrue + curFalse), threshold[i]);
        liftChart[i] = new EvaluationCurvePoint(1.0 * (curTrue + curFalse) / total, curTrue, th
reshold[i]);
    }

// 以我们例子，得到
curTrue = 3
curFalse = 2

threshold = {double[7]@9349}
 0 = 0.0
 1 = 0.9
 2 = 0.8
 3 = 0.7500000000000001
 4 = 0.7000000000000001
 5 = 0.6000000000000001
 6 = 0.5

rocCurve = {EvaluationCurvePoint[7]@9315}
 1 = {EvaluationCurvePoint@9440}
  x = 0.0
  y = 0.3333333333333333
  p = 0.9
 2 = {EvaluationCurvePoint@9448}
  x = 0.0
  y = 0.6666666666666666
  p = 0.8
 3 = {EvaluationCurvePoint@9449}
  x = 0.5
  y = 0.6666666666666666
  p = 0.7500000000000001
 4 = {EvaluationCurvePoint@9450}
  x = 0.5
  y = 1.0
  p = 0.7000000000000001
 5 = {EvaluationCurvePoint@9451}
  x = 1.0
  y = 1.0
  p = 0.6000000000000001
 6 = {EvaluationCurvePoint@9452}
  x = 1.0
  y = 1.0
  p = 0.5

recallPrecisionCurve = {EvaluationCurvePoint[7]@9320}
 1 = {EvaluationCurvePoint@9444}
  x = 0.3333333333333333
  y = 1.0
  p = 0.9
 2 = {EvaluationCurvePoint@9453}
  x = 0.6666666666666666
  y = 1.0
```

```
  p = 0.8
 3 = {EvaluationCurvePoint@9454}
  x = 0.6666666666666666
  y = 0.6666666666666666
  p = 0.7500000000000001
 4 = {EvaluationCurvePoint@9455}
  x = 1.0
  y = 0.75
  p = 0.7000000000000001
 5 = {EvaluationCurvePoint@9456}
  x = 1.0
  y = 0.6
  p = 0.6000000000000001
 6 = {EvaluationCurvePoint@9457}
  x = 1.0
  y = 0.6
  p = 0.5

liftChart = {EvaluationCurvePoint[7]@9325}
 1 = {EvaluationCurvePoint@9458}
  x = 0.2
  y = 1.0
  p = 0.9
 2 = {EvaluationCurvePoint@9459}
  x = 0.4
  y = 2.0
  p = 0.8
 3 = {EvaluationCurvePoint@9460}
  x = 0.6
  y = 2.0
  p = 0.7500000000000001
 4 = {EvaluationCurvePoint@9461}
  x = 0.8
  y = 3.0
  p = 0.7000000000000001
 5 = {EvaluationCurvePoint@9462}
  x = 1.0
  y = 3.0
  p = 0.6000000000000001
 6 = {EvaluationCurvePoint@9463}
  x = 1.0
  y = 3.0
  p = 0.5

data = {ConfusionMatrix[7]@9339}
 0 = {ConfusionMatrix@9486}
  longMatrix = {LongMatrix@9488}
   matrix = {long[2][]@9491}
    0 = {long[2]@9492}
     0 = 0
     1 = 0
    1 = {long[2]@9493}
     0 = 3
     1 = 2
   rowNum = 2
   colNum = 2
  labelCnt = 2
  total = 5
  actualLabelFrequency = {long[2]@9489}
   0 = 3
```

```
   1 = 2
  predictLabelFrequency = {long[2]@9490}
   0 = 0
   1 = 5
  tpCount = 2.0
  tnCount = 2.0
  fpCount = 3.0
  fnCount = 3.0
 1 = {ConfusionMatrix@9435}
  longMatrix = {LongMatrix@9469}
   matrix = {long[2][]@9472}
    0 = {long[2]@9474}
     0 = 1
     1 = 0
    1 = {long[2]@9475}
     0 = 2
     1 = 2
   rowNum = 2
   colNum = 2
  labelCnt = 2
  total = 5
  actualLabelFrequency = {long[2]@9470}
   0 = 3
   1 = 2
  predictLabelFrequency = {long[2]@9471}
   0 = 1
   1 = 4
  tpCount = 3.0
  tnCount = 3.0
  fpCount = 2.0
  fnCount = 2.0
  ......

    threshold[0] = 1.0;
    data[0] = new ConfusionMatrix(new long[][] {{0, 0}, {totalTrue, totalFalse}});
    rocCurve[0] = new EvaluationCurvePoint(0, 0, threshold[0]);
    recallPrecisionCurve[0] = new EvaluationCurvePoint(0, recallPrecisionCurve[1].getY(), thres
hold[0]);
    liftChart[0] = new EvaluationCurvePoint(0, 0, threshold[0]);

    return Tuple3.of(data, threshold, new EvaluationCurve[] {new EvaluationCurve(rocCurve),
        new EvaluationCurve(recallPrecisionCurve), new EvaluationCurve(liftChart)});
}
```

3.2.4 计算混淆矩阵

这里再给大家讲讲混淆矩阵如何计算，这里思路比较绕。

3.2.4.1 原始矩阵

调用之处是：

```
// 调用之处
data[i] = new ConfusionMatrix(
        new long[][] {{curTrue, curFalse}, {totalTrue - curTrue, totalFalse - curFalse}});
// 调用时候各种赋值
i = 1
index = 90000
totalTrue = 3
totalFalse = 2
```

```
curTrue = 1
curFalse = 0
```

得到原始矩阵，以下都有cur，说明只针对当前点来说。

curTrue = 1	curFalse = 0
totalTrue - curTrue = 2	totalFalse - curFalse = 2

3.2.4.2 计算标签

后续ConfusionMatrix计算中，由此可以得到

```
actualLabelFrequency = longMatrix.getColSums();
predictLabelFrequency = longMatrix.getRowSums();

actualLabelFrequency = {long[2]@9322}
 0 = 3
 1 = 2
predictLabelFrequency = {long[2]@9323}
 0 = 1
 1 = 4
```

可以看出来，Alink算法认为：每列的sum和实际标签有关；每行sum和预测标签有关。

得到新矩阵如下

			predictLabelFrequency
	curTrue = 1	curFalse = 0	1 = curTrue + curFalse
	totalTrue - curTrue = 2	totalFalse - curFalse = 2	4 = total - curTrue - curFalse
actualLabelFrequency	3 = totalTrue	2 = totalFalse	

后续计算将要基于这些来计算：

计算中就用到longMatrix 对角线上的数据，即longMatrix(0)(0)和 longMatrix(1)(1)。一定要注意，这里考虑的都是 **当前状态 (画重点强调)**。

longMatrix(0)(0) ： curTrue

longMatrix(1)(1) ： totalFalse - curFalse

totalFalse ： (TN + FN)

totalTrue ： (TP + FP)

```
double numTrueNegative(Integer labelIndex) {
  // labelIndex为 0 时候, return 1 + 5 - 1 - 3 = 2;
  // labelIndex为 1 时候, return 2 + 5 - 4 - 2 = 1;
        return null == labelIndex ? tnCount : longMatrix.getValue(labelIndex, labelIndex) + total - predictLabelFrequency[labelIndex] - actualLabelFrequency[labelIndex];
}
```

```
double numTruePositive(Integer labelIndex) {
  // labelIndex为 0 时候, return 1; 这个是 curTrue, 就是真实标签是True, 判别也是True。是TP
  // labelIndex为 1 时候, return 2; 这个是 totalFalse - curFalse, 总判别错 - 当前判别错。这就意味着"本来判别错了但是当前没有发现", 所以认为在当前状态下, 这也算是TP
        return null == labelIndex ? tpCount : longMatrix.getValue(labelIndex, labelIndex);
}

double numFalseNegative(Integer labelIndex) {
  // labelIndex为 0 时候, return 3 - 1;
  // actualLabelFrequency[0] = totalTrue。所以return totalTrue - curTrue, 即当前"全部正确"中没有"判别为正确", 这个就可以认为是"判别错了且判别为负"
  // labelIndex为 1 时候, return 2 - 2;
  // actualLabelFrequency[1] = totalFalse。所以return totalFalse - ( totalFalse - curFalse )  = curFalse
        return null == labelIndex ? fnCount : actualLabelFrequency[labelIndex] - longMatrix.getValue(labelIndex, labelIndex);
}

double numFalsePositive(Integer labelIndex) {
  // labelIndex为 0 时候, return 1 - 1;
  // predictLabelFrequency[0] = curTrue + curFalse。
  // 所以 return = curTrue + curFalse - curTrue = curFalse = current( TN + FN ) 这可以认为是判断错了实际是正确标签
  // labelIndex为 1 时候, return 4 - 2;
  // predictLabelFrequency[1] = total - curTrue - curFalse。
  // 所以 return = total - curTrue - curFalse - (totalFalse - curFalse) = totalTrue - curTrue = ( TP + FP ) - currentTP = currentFP
        return null == labelIndex ? fpCount : predictLabelFrequency[labelIndex] - longMatrix.getValue(labelIndex, labelIndex);
}

// 最后得到
tpCount = 3.0
tnCount = 3.0
fpCount = 2.0
fnCount = 2.0
```

3.2.4.3 具体代码

```
// 具体计算
public ConfusionMatrix(LongMatrix longMatrix) {

longMatrix = {LongMatrix@9297}
  0 = {long[2]@9324}
   0 = 1
   1 = 0
  1 = {long[2]@9325}
   0 = 2
   1 = 2

    this.longMatrix = longMatrix;
    labelCnt = this.longMatrix.getRowNum();
    // 这里就是计算
    actualLabelFrequency = longMatrix.getColSums();
    predictLabelFrequency = longMatrix.getRowSums();

actualLabelFrequency = {long[2]@9322}
 0 = 3
 1 = 2
predictLabelFrequency = {long[2]@9323}
```

```
 0 = 1
 1 = 4
labelCnt = 2
total = 5

    total = longMatrix.getTotal();
    for (int i = 0; i < labelCnt; i++) {
        tnCount += numTrueNegative(i);
        tpCount += numTruePositive(i);
        fnCount += numFalseNegative(i);
        fpCount += numFalsePositive(i);
    }
}
```

0x04 流处理

4.1 示例

Alink原有python示例代码中，Stream部分是没有输出的，因为MemSourceStreamOp没有和时间相关联，而Alink中没有提供基于时间的StreamOperator，所以只能自己仿照MemSourceBatchOp写了一个。虽然代码有些丑，但是至少可以提供输出，这样就能够调试。

4.1.1 主类

```
public class EvalBinaryClassExampleStream {

    AlgoOperator getData(boolean isBatch) {
        Row[] rows = new Row[]{
                Row.of("prefix1", "{\"prefix1\": 0.9, \"prefix0\": 0.1}")
        };
        String[] schema = new String[]{"label", "detailInput"};
        if (isBatch) {
            return new MemSourceBatchOp(rows, schema);
        } else {
            return new TimeMemSourceStreamOp(rows, schema, new EvalBinaryStreamSource());
        }
    }

    public static void main(String[] args) throws Exception {
        EvalBinaryClassExampleStream test = new EvalBinaryClassExampleStream();
        StreamOperator streamData = (StreamOperator) test.getData(false);
        StreamOperator sOp = new EvalBinaryClassStreamOp()
                .setLabelCol("label")
                .setPredictionDetailCol("detailInput")
                .setTimeInterval(1)
                .linkFrom(streamData);
        sOp.print();
        StreamOperator.execute();
    }
}
```

4.1.2 TimeMemSourceStreamOp

这个是我自己炮制的。借鉴了MemSourceStreamOp。

```
public final class TimeMemSourceStreamOp extends StreamOperator<TimeMemSourceStreamOp> {

    public TimeMemSourceStreamOp(Row[] rows, String[] colNames, EvalBinaryStrSource source) {
        super(null);
        init(source, Arrays.asList(rows), colNames);
```

```
    }

    private void init(EvalBinaryStreamSource source, List <Row> rows, String[] colNames) {
        Row first = rows.iterator().next();
        int arity = first.getArity();
        TypeInformation <?>[] types = new TypeInformation[arity];

        for (int i = 0; i < arity; ++i) {
            types[i] = TypeExtractor.getForObject(first.getField(i));
        }

        init(source, colNames, types);
    }

    private void init(EvalBinaryStreamSource source, String[] colNames, TypeInformation <?>[] c
olTypes) {
        DataStream <Row> dastr = MLEnvironmentFactory.get(getMLEnvironmentId())
                .getStreamExecutionEnvironment().addSource(source);
        StringBuilder sbd = new StringBuilder();
        sbd.append(colNames[0]);

        for (int i = 1; i < colNames.length; i++) {
            sbd.append(",").append(colNames[i]);
        }
        this.setOutput(dastr, colNames, colTypes);
    }

    @Override
    public TimeMemSourceStreamOp linkFrom(StreamOperator<?>... inputs) {
        return null;
    }
}
```

4.1.3 Source

定时提供Row，加入了随机数，让概率有变化。

```
class EvalBinaryStreamSource extends RichSourceFunction[Row] {

  override def run(ctx: SourceFunction.SourceContext[Row]) = {
    while (true) {
      val rdm = Math.random() // 这里加入了随机数，让概率有变化
      val rows: Array[Row] = Array[Row](
        Row.of("prefix1", "{\"prefix1\": " + rdm + ", \"prefix0\": " + (1-rdm) + "}"),
        Row.of("prefix1", "{\"prefix1\": 0.8, \"prefix0\": 0.2}"),
        Row.of("prefix1", "{\"prefix1\": 0.7, \"prefix0\": 0.3}"),
        Row.of("prefix0", "{\"prefix1\": 0.75, \"prefix0\": 0.25}"),
        Row.of("prefix0", "{\"prefix1\": 0.6, \"prefix0\": 0.4}"))
      for(row <- rows) {
        println(s"当前值：$row")
        ctx.collect(row)
      }
      Thread.sleep(1000)
    }
  }

  override def cancel() = ???
}
```

4.2 BaseEvalClassStreamOp

Alink流处理类是 EvalBinaryClassStreamOp，主要工作在其基类 BaseEvalClassStreamOp，所以我们重点看后者。

```
public class BaseEvalClassStreamOp<T extends BaseEvalClassStreamOp<T>> extends StreamOperator<T
> {
    @Override
    public T linkFrom(StreamOperator<?>... inputs) {
        StreamOperator<?> in = checkAndGetFirst(inputs);
        String labelColName = this.get(MultiEvaluationStreamParams.LABEL_COL);
        String positiveValue = this.get(BinaryEvaluationStreamParams.POS_LABEL_VAL_STR);
        Integer timeInterval = this.get(MultiEvaluationStreamParams.TIME_INTERVAL);

        ClassificationEvaluationUtil.Type type = ClassificationEvaluationUtil.judgeEvaluationTy
pe(this.getParams());

        DataStream<BaseMetricsSummary> statistics;

        switch (type) {
            case PRED_RESULT: {
              ......
            }
            case PRED_DETAIL: {
                String predDetailColName = this.get(MultiEvaluationStreamParams.PREDICTION_DETA
IL_COL);
                //
                PredDetailLabel eval = new PredDetailLabel(positiveValue, binary);
                // 获取输入数据，重点是timeWindowAll
                statistics = in.select(new String[] {labelColName, predDetailColName})
                    .getDataStream()
                    .timeWindowAll(Time.of(timeInterval, TimeUnit.SECONDS))
                    .apply(eval);
                break;
            }
        }
        // 把各个窗口的数据累积到 totalStatistics，注意，这里是新变量了。
        DataStream<BaseMetricsSummary> totalStatistics = statistics
            .map(new EvaluationUtil.AllDataMerge())
            .setParallelism(1); // 并行度设置为1

        // 基于两种 bins 计算&序列化，得到当前的 statistics
        DataStream<Row> windowOutput = statistics.map(
            new EvaluationUtil.SaveDataStream(ClassificationEvaluationUtil.WINDOW.f0));
        // 基于bins计算&序列化，得到累积的 totalStatistics
        DataStream<Row> allOutput = totalStatistics.map(
            new EvaluationUtil.SaveDataStream(ClassificationEvaluationUtil.ALL.f0));

        // "当前" 和 "累积" 做联合，最终返回
        DataStream<Row> union = windowOutput.union(allOutput);

        this.setOutput(union,
            new String[] {ClassificationEvaluationUtil.STATISTICS_OUTPUT, DATA_OUTPUT},
            new TypeInformation[] {Types.STRING, Types.STRING});

        return (T)this;
    }
}
```

具体业务是：

- PredDetailLabel 会进行去重标签名字 和 累积计算混淆矩阵所需数据
    - buildLabelIndexLabelArray 去重 "labels名字"，然后给每一个label一个ID，最后结果是一个<labels, ID>Map。
    - getDetailStatistics 遍历 rows 数据，提取每一个item（比如 "prefix1,{"prefix1": 0.8, "prefix0": 0.2}"），然后通过updateBinaryMetricsSummary累积计算混淆矩阵所需数据。
- 根据标签从Window中获取数据 statistics = in.select().getDataStream().timeWindowAll() .apply(eval);
- EvaluationUtil.AllDataMerge 把各个窗口的数据累积到 totalStatistics 。
- 得到windowOutput -------- EvaluationUtil.SaveDataStream，对"当前数据statistics"做处理。实际业务在BinaryMetricsSummary.toMetrics，即基于bin的信息计算，然后存储到params，并序列化返回Row。
    - extractMatrixThreCurve函数取出非空的bins，据此计算出ConfusionMatrix array（混淆矩阵），threshold array, rocCurve/recallPrecisionCurve/LiftChart.
    - 依据曲线内容计算并且存储 AUC/PRC/KS
    - 对生成的rocCurve/recallPrecisionCurve/LiftChart输出进行抽样
    - 依据抽样后的输出存储 RocCurve/RecallPrecisionCurve/LiftChar
    - 存储正例样本的度量指标
    - 存储Logloss
    - Pick the middle point where threshold is 0.5.
- 得到allOutput -------- EvaluationUtil.SaveDataStream，对"累积数据totalStatistics"做处理。
    - 详细处理流程同windowOutput。
- windowOutput 和 allOutput 做联合。最终返回 DataStream union = windowOutput.union(allOutput);

4.2.1 PredDetailLabel

```
static class PredDetailLabel implements AllWindowFunction<Row, BaseMetricsSummary, TimeWindow>
{
    @Override
    public void apply(TimeWindow timeWindow, Iterable<Row> rows, Collector<BaseMetricsSummary>
collector) throws Exception {
        HashSet<String> labels = new HashSet<>();
        // 首先还是获取 labels 名字
        for (Row row : rows) {
            if (EvaluationUtil.checkRowFieldNotNull(row)) {
                labels.addAll(EvaluationUtil.extractLabelProbMap(row).keySet());
                labels.add(row.getField(0).toString());
            }
        }
labels = {HashSet@9757}  size = 2
 0 = "prefix1"
 1 = "prefix0"
        // 之前介绍过，buildLabelIndexLabelArray 去重 "labels名字"，然后给每一个label一个ID，最后结果是
一个<labels, ID>Map。
        // getDetailStatistics 遍历 rows 数据，累积计算混淆矩阵所需数据 ( "TP + FN"  /  "TN + FP")。
        if (labels.size() > 0) {
            collector.collect(
                getDetailStatistics(rows, binary, buildLabelIndexLabelArray(labels, binary, pos
itiveValue)));
        }
    }
}
```

4.2.2 AllDataMerge

EvaluationUtil.AllDataMerge 把各个窗口的数据累积

```java
/**
 * Merge data from different windows.
 */
public static class AllDataMerge implements MapFunction<BaseMetricsSummary, BaseMetricsSummary>
 {
    private BaseMetricsSummary statistics;
    @Override
    public BaseMetricsSummary map(BaseMetricsSummary value) {
        this.statistics = (null == this.statistics ? value : this.statistics.merge(value));
        return this.statistics;
    }
}
```

4.2.3 SaveDataStream

SaveDataStream具体调用的函数之前批处理介绍过，实际业务在BinaryMetricsSummary.toMetrics，即基于bin的信息计算，存储到params。

这里与批处理不同的是直接就把"构建出的度量信息"返回给用户。

```java
public static class SaveDataStream implements MapFunction<BaseMetricsSummary, Row> {
    @Override
    public Row map(BaseMetricsSummary baseMetricsSummary) throws Exception {
        BaseMetricsSummary metrics = baseMetricsSummary;
        BaseMetrics baseMetrics = metrics.toMetrics();
        Row row = baseMetrics.serialize();
        return Row.of(funtionName, row.getField(0));
    }
}

// 最后得到的 row 其实就是最终返回给用户的度量信息
row = {Row@10008} "{"PRC":"0.9164636268708667","SensitivityArray":"[0.38461538461538464,0.6923076923076923,0.6923076923076923,1.0,1.0,1.0]","ConfusionMatrix":"[[13,8],[0,0]]","MacroRecall":"0.5","MacroSpecificity":"0.5","FalsePositiveRateArray":"[0.0,0.0,0.5,0.5,1.0,1.0]" ...... 还有很多其他的
```

4.2.4 Union

```java
DataStream<Row> windowOutput = statistics.map(
    new EvaluationUtil.SaveDataStream(ClassificationEvaluationUtil.WINDOW.f0));
DataStream<Row> allOutput = totalStatistics.map(
    new EvaluationUtil.SaveDataStream(ClassificationEvaluationUtil.ALL.f0));

DataStream<Row> union = windowOutput.union(allOutput);
```

最后返回两种统计数据

4.2.4.1 allOutput


```
all|{"PRC":"0.7341146115890359","SensitivityArray":"[0.3333333333333333,0.3333333333333333,0.6666666666666666,0.7333333333333333,0.8,0.8,0.8666666666666667,0.8666666666666667,0.9333333333333333,1.0]","ConfusionMatrix":"[[13,10],[2,0]]","MacroRecall":"0.43333333333333335","MacroSpecificity":"0.43333333333333335","FalsePositiveRateArray":"[0.0,0.5,0.5,0.5,0.5,1.0,1.0,1.0,1.0,1.0]","TruePositiveRateArray":"[0.3333333333333333,0.3333333333333333,0.6666666666666666,0.7333333333333333,0.8,0.8,0.8666666666666667,0.8666666666666667,0.9333333333333333,1.0]","AUC":"0.5666666666666667","MacroAccuracy":"0.52", ......
```


4.2.4.2 windowOutput


```
window|{"PRC":"0.7638888888888888","SensitivityArray":"[0.3333333333333333,0.3333333333333333,0.6666666666666666,1.0,1.0,1.0]","ConfusionMatrix":"[[3,2],[0,0]]","MacroRecall":"0.5","MacroSpe
```

```
cificity":"0.5","FalsePositiveRateArray":"[0.0,0.5,0.5,0.5,1.0,1.0]","TruePositiveRateArray":"[
0.3333333333333333,0.3333333333333333,0.6666666666666666,1.0,1.0,1.0]","AUC":"0.666666666666666
6","MacroAccuracy":"0.6","RecallArray":"[0.3333333333333333,0.3333333333333333,0.66666666666666
66,1.0,1.0,1.0]","KappaArray":"[0.28571428571428564,-0.15384615384615377,0.1666666666666666,0.5
454545454545455,0.0,0.0]","MicroFalseNegativeRate":"0.4","WeightedRecall":"0.6","WeightedPrecis
ion":"0.36","Recall":"1.0","MacroPrecision":"0.3",......
```

0xFF 参考

[[白话解析] 通过实例来梳理概念 ： 准确率 (Accuracy)、精准率(Precision)、召回率(Recall) 和 F值(F-Measure)](