

华中科技大学

课程报告

大数据处理与  
MapReduce 编程模型

专业班级 计算机

学 号

姓 名

指导老师

2024 年 1 月 14 日

计算机科学与技术学院

## 目 录

<b>1 实验目的与要求</b> .....	<b>1</b>
1.1 实验目的.....	1
1.2 实验要求.....	1
<b>2 贝叶斯分类器理论介绍</b> .....	<b>3</b>
2.1 贝叶斯定理.....	3
2.2 贝叶斯分类器.....	3
2.3 朴素贝叶斯分类器.....	5
2.4 贝叶斯分类器在本实验的应用.....	6
<b>3 贝叶斯分类器训练的 MapReduce 算法设计</b> .....	<b>8</b>
3.1 系统框架实现.....	8
3.2 Job1: ClassCount 实现.....	9
3.3 Job2: WordCount 实现.....	12
3.4 Job3: Predict 实现.....	14
<b>4 实验与结果分析</b> .....	<b>19</b>
4.1 实验环境.....	19
4.2 实验数据集.....	19
4.3 实验运行分析.....	21
4.4 实验结果分析.....	24

## 1 实验目的与要求

### 1.1 实验目的

本实验旨在通过实现一个基于贝叶斯分类算法的 MapReduce 框架程序，完成大规模文档分类任务。通过本实验，学生能够掌握以下技能：

#### 1. 贝叶斯分类模型的训练与实现

学习如何使用 MapReduce 框架并行化处理文档数据，基于贝叶斯定理训练文档分类模型。通过计算每个类别的先验概率和每个单词在各类别下的条件概率，构建能够准确预测文档类别的分类器。

#### 2. 测试集分类与模型验证

使用训练得到的贝叶斯分类模型对测试集文档进行分类，并输出每个文档的预测类别。通过实现不同的测试方式，学生将了解如何将训练好的模型应用于新数据，并检查分类器的准确性。

#### 3. 分类模型的评估

学会通过计算精准度（Precision）、召回率（Recall）和 F1 值等指标，评估分类模型在实际任务中的性能。通过这些指标对分类器的效果进行量化分析，进而为进一步优化模型提供参考。

### 1.2 实验要求

#### 1. 训练模型的实现

实现一个 MapReduce 作业，用于训练贝叶斯分类模型。训练过程需要从文档数据集中提取每个类别的文档数量、每个类别内单词的频率等信息，并计算先验概率和条件概率。在训练过程中，应考虑数据的分布和计算的效率。

#### 2. 测试集的分类

通过使用训练好的模型对测试集文档进行分类，可以选择实现一个基于 MapReduce 的分类测试程序，或使用传统的单机 Java 程序进行分类。无论采用哪种方式，最终需要输出每个测试文档的预测类别，并与真实类别进行对比。

### 3. 性能评估

在测试过程中，计算分类结果的 Precision、Recall 和 F1 值。要求使用适当的公式和方法计算这些指标，并根据指标评估模型的性能。具体公式为：

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

$$\text{F1} = 2 * (\text{Precision} * \text{Recall}) / (\text{Precision} + \text{Recall})$$

其中，TP 为真正例，FP 为假正例，FN 为假负例，TN 为真负例。需要在报告中展示这些指标的计算过程，并对结果进行分析和讨论。

### 4. 结果分析

完成分类测试和性能评估后，需要对分类结果进行分析，讨论分类模型的优缺点，并提出可能的优化方案。例如，可以讨论训练数据的质量、模型的泛化能力以及在不同类别文档上的表现差异等问题。

## 2 贝叶斯分类器理论介绍

贝叶斯分类器（Bayes Classifier）是一种基于贝叶斯定理的统计分类方法。它利用条件概率模型，通过对训练数据集的学习，构建一个模型来预测新数据的类别。贝叶斯分类器在机器学习、自然语言处理和数据挖掘等领域广泛应用，尤其在文本分类、垃圾邮件过滤、情感分析等任务中表现出色。

### 2.1 贝叶斯定理

贝叶斯定理是贝叶斯分类器的基础，公式如下：

$$P(C_k|X) = \frac{P(X|C_k) \cdot P(C_k)}{P(X)}$$

其中：

$P(C_k|X)$  是给定观测数据  $X$  后，类别  $C_k$  的后验概率。

$P(X|C_k)$  是在类别  $C_k$  条件下，观测数据  $X$  的似然函数，即给定类别，特征出现的概率。

$P(C_k)$  是类别  $C_k$  的先验概率，即类别在整个数据集中出现的概率。

$P(X)$  是观测数据  $X$  的证据，通常是所有类别的似然函数与先验概率的加权平均。

贝叶斯定理的核心思想是，通过已知的训练数据，更新对各类别的信念（后验概率）。在分类任务中，贝叶斯分类器通过计算每个类别的后验概率，选择具有最高后验概率的类别作为预测结果。

### 2.2 贝叶斯分类器

贝叶斯分类器是用于分类的贝叶斯网络，它是概率分类器家族中的基础分类器。贝叶斯分类器的主要优势在于其模型的可解释性和较高的分类精度，但由于其模型结构较复杂，因此计算量较大。图 2-1 显示了贝叶斯分类器的一个典型结构，其中包含了  $n$  个属性节点  $x_1, x_2, \dots, x_n$  和一个类节点  $C$ 。

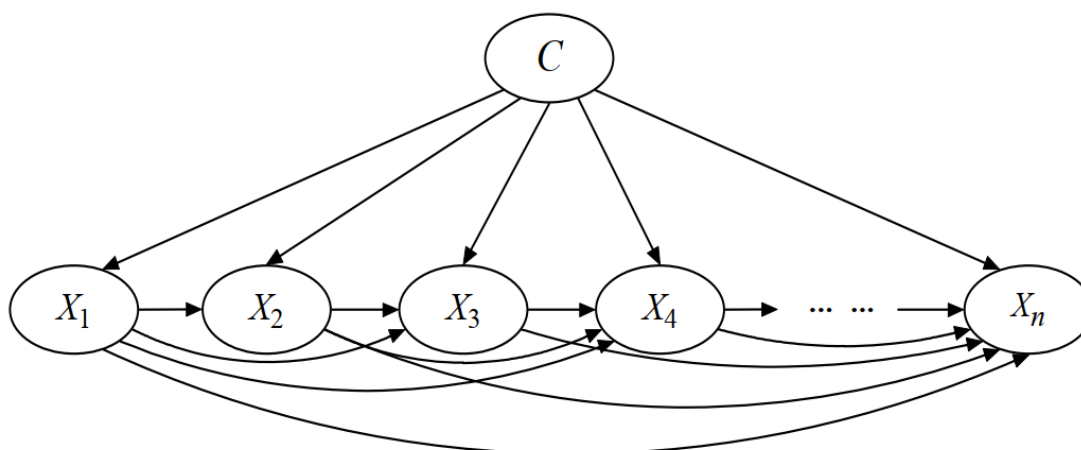


图 2-1 贝叶斯分类器结构

贝叶斯分类器的目标是基于训练数据估计类别的先验概率和每个特征在各类别条件下的概率，并根据这些概率进行分类预测。贝叶斯分类器的主要步骤包括以下几个方面：

### 1. 计算类别的先验概率

贝叶斯分类器首先计算每个类别在训练数据中出现的频率。先验概率  $P(C_k)$  是每个类别的比例，通常通过训练集中的类别标签来计算。

### 2. 计算条件概率

对于每个特征  $x_i$ ，计算在某个类别  $C_k$  下  $x_i$  出现的概率  $P(x_i|C_k)$ ，这就是条件概率。假设特征之间是独立的（即条件独立假设），则联合概率可以表示为特征的条件概率的乘积：

$$P(X|C_k) = P(x_1|C_k) \cdot P(x_2|C_k) \dots P(x_n|C_k)$$

这种假设极大简化了计算过程，因此称为“朴素贝叶斯”（Naive Bayes）。

### 3. 计算后验概率并分类

对于每个类别  $C_k$ ，通过贝叶斯定理计算后验概率：

$$P(C_k|X) = \frac{P(X|C_k) \cdot P(C_k)}{P(X)}$$

然后选择具有最大后验概率的类别作为最终预测结果：

$$\hat{C} = \arg \max_{C_k} P(C_k|X)$$

由于  $P(X)$  对所有类别是相同的，因此在实际应用中通常忽略  $P(X)$ ，仅通过  $P(X|C_k) \cdot P(C_k)$  来计算后验概率，并选择最大值。

贝叶斯分类器的主要优点有：

- **计算效率高**：贝叶斯分类器计算简单，不需要大量的计算资源，适合大规模数据集。
- **处理缺失数据**：贝叶斯分类器能较好地处理缺失数据，特别是在某些特征缺失时，可以依靠其他特征进行预测。
- **鲁棒性强**：贝叶斯分类器即使在特征之间存在一定依赖关系时，也能提供合理的分类结果。

贝叶斯分类器的主要缺点有：

- **特征独立假设**：贝叶斯分类器假设所有特征在给定类别下是独立的，但在许多实际应用中，特征往往是相关的，这可能会影响分类器的表现。
- **类别条件概率的估计**：在数据稀疏时，某些类别的条件概率可能为零，导致模型无法处理这些特征。通常使用拉普拉斯平滑技术来解决这个问题。

## 2.3 朴素贝叶斯分类器

与贝叶斯分类器的复杂性相比，朴素贝叶斯分类器是概率分类器中最简单的一种。它基于贝叶斯定理，并在此基础上做出了一个重要的假设：假设所有特征在给定类别的条件下是条件独立的。也就是说，属性变量之间的依赖关系被忽略，所有特征对类别的贡献是独立的。这个假设使得朴素贝叶斯分类器的计算大大简化，并且提高了其在实际应用中的效率。

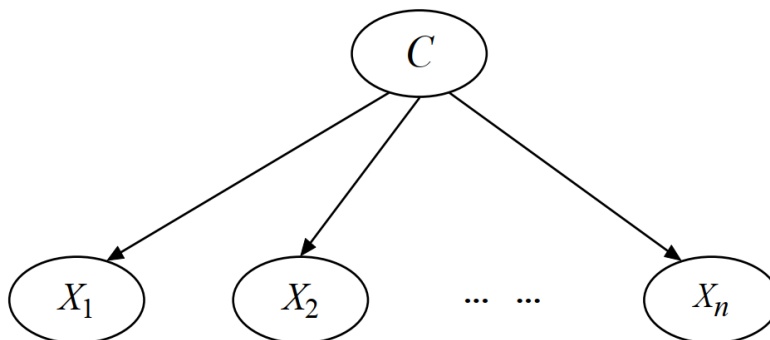


图 2-2 朴素贝叶斯分类器结构

朴素贝叶斯分类器的结构可以用一个简单的星形结构来表示，如图 2-2 所示。其中每个特征节点只有一个父节点——类别节点  $C$ 。在这个结构中，当类别  $C$

给定时, 特征  $x_1, x_2, \dots, x_n$  条件独立。

朴素贝叶斯分类器的基本步骤包括:

## 1. 训练阶段

在训练阶段, 模型计算每个类别的先验概率  $P(C_k)$ , 并计算每个特征在每个类别下的条件概率  $P(x_i|C_k)$ 。

## 2. 预测阶段

在预测阶段, 对于给定的文档 (或数据点), 计算每个类别的后验概率, 并选择概率最大的一类作为预测结果。

朴素贝叶斯分类器的优点在于其结构简单、计算效率高, 特别适合用于文本分类等高维数据的分类任务。尽管朴素贝叶斯分类器做出了条件独立的假设, 在许多实际应用中, 它仍然能够取得相当高的分类准确率。特别是在噪声较多或者数据不完全的情况下, 朴素贝叶斯分类器表现出了较强的鲁棒性。

## 2.4 贝叶斯分类器在本实验的应用

在本实验中, 贝叶斯分类器用于文本分类任务, 具体分为训练阶段和测试阶段。训练阶段通过计算训练数据中的先验概率和条件概率, 建立贝叶斯分类模型; 测试阶段利用该模型对测试文档进行分类, 并评估分类器的性能。

### 1. 训练阶段

训练阶段的目标是通过训练数据计算每个类别的先验概率  $P(C_k)$  和文档中词条的条件概率  $P(X|C_k)$ 。根据贝叶斯定理, 后验概率  $P(C_k|X)$  与先验概率和条件概率的乘积成正比, 因此只需最大化  $P(X|C_k) \cdot P(C_k)$  即可确定文档  $X$  的最可能类别  $C_k$ 。

先验概率计算。先验概率  $P(C_k)$  表示文档属于类别  $C_k$  的概率, 计算公式为:

$$P(C_k) = \frac{N(C_k)}{N}$$

其中,  $N(C_k)$  是类别为  $C_k$  的文档数量,  $N$  是训练集中的总文档数。

条件概率计算。为了计算文档  $X$  属于类别  $C_k$  的条件概率  $P(X|C_k)$ , 假设



文档中每个词条的出现是相互独立的。基于这一假设，条件概率可以表示为：

$$P(X|C_k) = P(x_1|C_k) \cdot P(x_2|C_k) \dots P(x_n|C_k)$$

其中， $x_i$  是文档中的第  $i$  个词条， $n$  是文档  $X$  中的词条总数， $P(x_i|C_k)$  表示词条  $x_i$  在类别  $C_k$  中出现的条件概率。该条件概率计算为：

$$P(x_i|C_k) = \frac{N(x_i|C_k)}{N(C_k)}$$

其中， $N(x_i|C_k)$  表示类别  $C_k$  的文档中词条  $x_i$  出现的次数。

## 2. 测试阶段

在测试阶段，贝叶斯分类器利用训练阶段计算的先验概率和条件概率，对测试文档进行分类。具体步骤如下：

对于每个测试文档  $x_i$ ，计算其属于每个类别  $C_k$  的后验概率  $P(C_k|X)$ ：

$$P(C_k|X) \propto P(X|C_k) \cdot P(C_k)$$

通过比较不同类别的后验概率值，选择最大值对应的类别作为文档  $X$  的预测类别。

对于每个测试文档，输出其预测类别，记录每个文档的分类结果。利用测试文档的真实类别标签，计算分类模型的精确率（Precision）、召回率（Recall）和 F1 值，评估模型的分类性能。

通过上述步骤，本实验利用贝叶斯分类器完成了对测试文档的分类，并通过性能指标对分类器的效果进行了全面评估。

## 3 贝叶斯分类器训练的 MapReduce 算法设计

### 3.1 系统框架实现

该项目实现了基于贝叶斯分类的文本分类算法，采用 Hadoop MapReduce 架构。项目的目录结构包括 jobs 和 utils 两个主要部分，其中 jobs 包含了分类和预测相关的 MapReduce 任务，而 utils 提供了相关的工具类和配置文件。系统代码文件框架如下所示。

```
project/src/main/java/com/xjh/  
├── jobs  
│   ├── ClassCount.java  
│   ├── Predict.java  
│   └── WordCount.java  
└── utils  
    ├── ClassInputFormat.java  
    ├── ClassRecordReader.java  
    ├── Config.java  
    ├── Predition.java  
    ├── WordInputFormat.java  
    └── WordRecordReader.java
```

#### 1. jobs 目录

该目录下包含了三个主要的 MapReduce 任务实现：ClassCount.java、Predict.java 和 WordCount.java。

**ClassCount.java:** 该任务用于统计训练集中的每个类别的文件数量，即计算各个类别的先验概率。输出格式为：<类名, 文件数量>。

**Predict.java:** 该任务负责根据先验概率和条件概率，对每个测试文件进行预测。输出预测结果，即每个文件的预测类别。

**WordCount.java:** 该任务统计每个类别下的单词频次，用于计算条件概率  $P(\text{单词}|\text{类名})$ 。

#### 2. utils 目录

该目录下包含了若干工具类和配置文件，主要用于支持 MapReduce 任务的实现。

**ClassInputFormat.java**：自定义的输入格式类，实现了 Hadoop 中 **InputFormat** 的相关功能，用于处理类别统计数据的输入。

**ClassRecordReader.java**：自定义的 **RecordReader** 类，负责读取 **ClassInputFormat** 的输入数据，解析文件并将每个类别的文档数量作为键值对输出。

**Config.java** 配置类，管理项目中的配置，如类别名、输入输出路径等，供其他类使用。

**Predition.java**：提供计算先验概率和条件概率的方法。通过训练集的统计数据，计算每个类别的概率，输出存储在 **priors** 和 **termProbability** 中。在 **Predict.java** 中被引用，计算预测结果时需要用到的先验概率和条件概率。

**WordInputFormat.java**：自定义的输入格式类，处理单词频次统计数据的输入流，并为 **WordRecordReader** 提供支持。

**WordRecordReader.java**：自定义的 **RecordReader** 类，负责读取 **WordInputFormat** 的输入数据，解析单词及其频次，并输出到 **Mapper**。

下面分别介绍 **ClassCount**、**Predict** 和 **WordCount** 三个 job 的具体实现。

## 3.2 Job1: ClassCount 实现

第一个 MapReduce 作业 **ClassCount** 的主要任务是统计每种类别（类别名称对应文档类型或文件夹名称）下的文档数量。每个类别的文档数量将用于计算贝叶斯分类器中的先验概率。先验概率是基于每个类别的文档数量对分类器进行权重计算的一部分，重要性在于帮助模型更好地理解每个类别在训练集中的相对频率。

任务输出是一个简单的键值对 <类别名称, 文件总数>，即每个类别下的文档数量。这些信息将在后续的任务中用于计算贝叶斯分类器的先验概率。

该任务使用了一个经典的 MapReduce 模式。具体流程如下：

**Mapper 阶段**：主要任务是将输入数据按照类别进行映射。假设输入的数据每行包含一个类别（例如 **AUSTR**）和该类别对应的文档计数。**Mapper** 不做额外处理，直接将类别和文档计数传递给 **Reducer**。输出格式为 <类别名称, 1>。

**Reducer 阶段：**汇总 Mapper 输出的键值对，将每个类别的文档数量进行相加，最终计算出每个类别的总文档数量。例如，某类别的文档数量可能是多个文件夹内文件的集合，因此 Reducer 会对来自不同 Mapper 的文档数量进行累加。输出格式为 <类别名称, 文件总数>。

该 job 的数据流如图 3-1 所示。输入数据是一个文件夹结构 CountryTrain，其中每个文件夹代表一个类别（例如，AUSTR、BRAZ、INDIA），文件夹内包含若干 txt 文件。输入的 HDFS 路径为 hdfs://host2:9000/CountryTrain。接着 ClassCountMapper 类负责将数据进一步处理，将每个类别名称和计数输出为 <类别名称, 1> 格式。在 Mapper 输出的数据进入 Reducer 之前，Hadoop 会进行 Shuffle 和 Sort 操作。ClassCountReducer 对每个类别进行聚合统计，将所有来自不同 Mapper 的同一类别的计数相加。每个类别的文档数量将通过对所有 1 值进行累加得到，最终输出每个类别的文档数量。最终的输出格式为<类别名称, 文件总数>，每个类别的文档数量即为该类别下所有文件夹内文件的总数。最终保存到指定的 HDFS 路径 hdfs://host2:9000/Result/ClassCount。

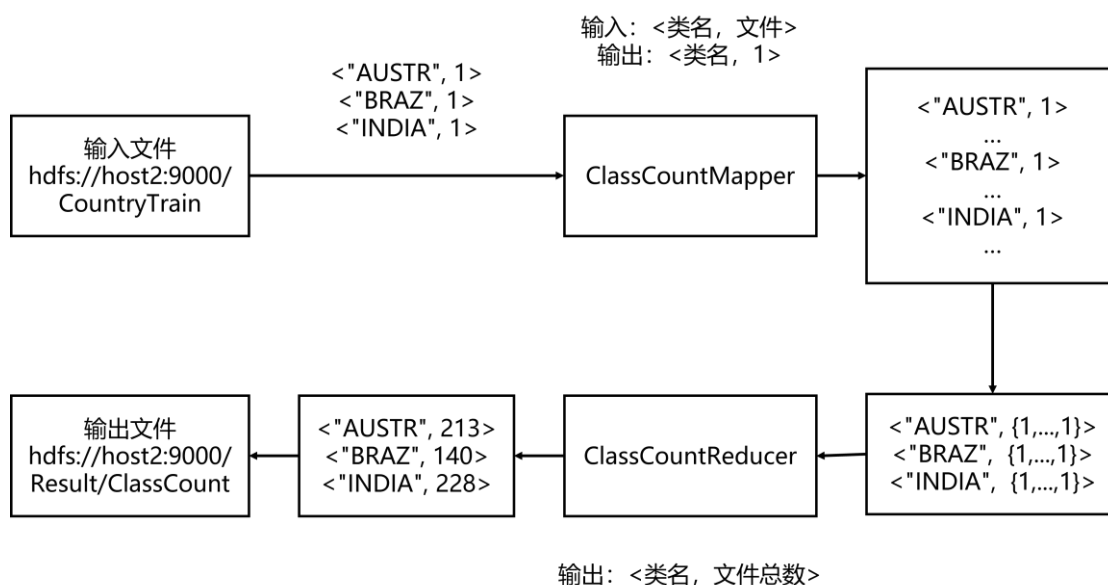


图 3-1 ClassCount 数据流图

该部分主要由 ClassCountMapper 类、ClassCountReducer 类、ClassRecordReader 类实现，下面分别介绍各个部分。

## 1. ClassCountMapper 类

Mapper 主要负责将输入数据处理成类别-计数的形式。由于输入数据是文件夹类型的数据, Mapper 需要处理每一个文档并输出 <类别名称, 1>, 这样每一个文件就会被赋予 1, 表示它属于一个类别。代码如下:

```
public static class ClassCountMapper extends Mapper<Text, IntWritable, Text, IntWritable> {
    @Override
    protected void map(Text key, IntWritable value, Context context) throws
        IOException, InterruptedException {
        context.write(key, value);
    }
}
```

## 2. ClassCountReducer 类

Reducer 的主要任务是对每个类别的文档数量进行汇总。具体来说, 它将对来自不同 Mapper 的同一类别的文档数量进行累加, 得到该类别的总文档数。代码如下:

```
public static class ClassCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable totalCount = new IntWritable();
    @Override
    protected void reduce(Text category, Iterable<IntWritable> counts, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable count : counts) {sum += count.get();}
        totalCount.set(sum);
        context.write(category, totalCount);
    }
}
```

## 3. ClassRecordReader 类

为了让 MapReduce 作业能够处理自定义格式的数据, 我们自定义了 ClassInputFormat 和 ClassRecordReader, 用来处理输入的目录数据并将其转化为 <类别名称, 1> 格式的键值对。

ClassInputFormat 继承自 FileInputFormat, 负责返回自定义的 RecordReader。

ClassRecordReader 负责从输入数据中读取目录的名称, 并返回每个文档的类别计数。关键代码如下:

```
public boolean nextKeyValue() throws IOException, InterruptedException {
    // 已经处理完所有文件
```

```

if (currentFileIndex >= files.length) { return false; }

// 初始化键和值
if (null == key) { key = new Text(); }
if (null == value) { value = new IntWritable(); }

// 设置当前键值对
key.set(className);
value.set(1); // 每个文件贡献 1 次计数

// 更新处理的文件索引
currentFileIndex++;
progressCounter++;
return true;
}
    
```

## 3.3 Job2: WordCount 实现

第二个 MapReduce 任务的目标是统计每个类别下的单词出现次数。每个输入文件包含某一类的多个单词，格式为 <类名, term1, term2, term3,...>。我们的任务是处理这些数据，输出每个单词在特定类别下出现的次数，输出格式为 <类名, 单词, 出现次数>。数据流如图 3-2 所示。

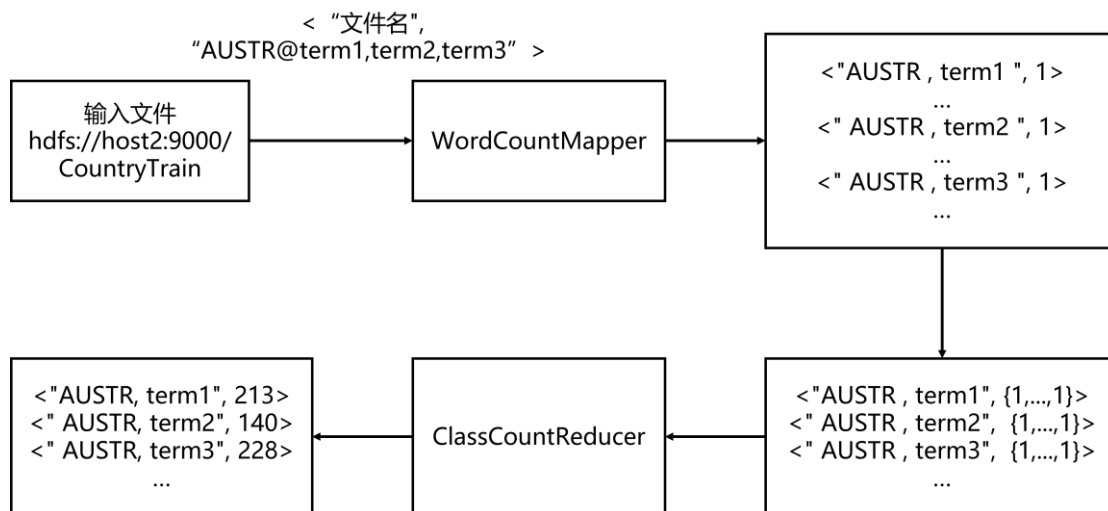


图 3-2 WordCount 数据流图

WordCount 的工作流程如下：

**Mapper 阶段：**从输入数据中提取类名和单词，并为每个单词生成一条键值对 <类名, 单词> -> 1，表示该单词在该类中出现一次。

**Reducer 阶段：**将 Mapper 的输出进行聚合，统计每个 <类名, 单词> 对应的

出现次数，最终输出 <类名, 单词, 出现次数>。

该部分主要由 WordCountMapper 类、WordCountReducer 类、ClassRecordReader 类实现，下面分别介绍各个部分。

## 1. WordCountMapper 类

Mapper 主要负责从输入数据中提取信息，并生成相应的键值对。输入的 key 是文件名（其实它没有实际作用），value 是一个字符串，包含了类名和逗号分隔的单词。使用 split("@") 分割类名和单词字符串，再用 split(",") 分割单词。然后对每个单词输出键值对 <类名, 单词> -> 1。代码如下：

```
public static class WordCountMapper extends Mapper<Text, Text, Text, IntWritable> {
    private IntWritable count = new IntWritable(1);

    @Override
    protected void map(Text key, Text value, Context context) throws IOException,
        InterruptedException {
        String[] split = value.toString().split("@");
        String className = split[0];
        String[] term = split[1].split(",");
        // 输出每个单词对应的计数（类名 + 单词）
        for (String s : term) {
            context.write(new Text(className + "," + s), count);
        }
    }
}
```

## 2. WordCountReducer 类

对于每个 <类名, 单词> 键，Reducer 会将其值（1）累加起来，最终输出每个单词的出现次数。关键代码如下：

```
public static class WordCountReducer extends Reducer<Text, IntWritable, Text,
    IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) { sum += val.get(); }
        result.set(sum);
        context.write(key, result);
    }
}
```

## 3. WordRecordReader 类

我们使用 WordInputFormat 和 WordRecordReader 两个类自定义输入格式，读取每个文件夹中的文件内容。其中 WordRecordReader 负责读取文件内容，并转换成 <文件名, 类名@单词 1,单词 2,...> 的格式。主要流程如下：

- **初始化：**initialize() 方法会从每个输入分片中读取路径信息，并获取该目录下的所有文件。
- **读取数据：**nextKeyValue() 方法会读取当前文件内容，并将其转化为 <文件名, 类名@单词列表> 的形式。
- **进度报告：**getProgress() 方法返回当前处理进度。

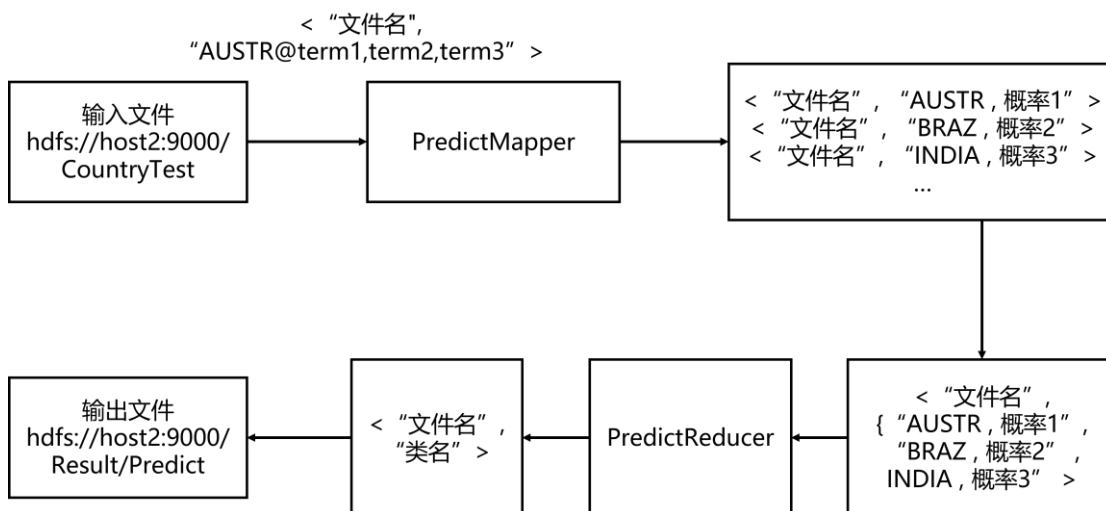


图 3-3 Predict 数据流图

## 3.4 Job3: Predict 实现

在本实验的第三个 Job 中，我们的目标是计算每个文件的预测类，并输出文件名及其对应的预测类别。具体来说，这个 Job 是基于贝叶斯分类的思想，通过计算文件属于各个类别的概率，最终选择概率最大的类别作为文件的预测类别。

该部分的输入数据格式为：<文件名, 类名@单词 1,单词 2,...>。“类名@单词 1,单词 2,...”表示文件的标记和单词列表。输出数据格式为：<文件名, 类名>，其中“类名”是预测的类别（比如 AUSTR、BRAZ、INDIA 等）。数据流如图 3-3 所示。



Predict 的工作流程如下：

**Mapper 阶段：**Mapper 会接收每个文件的内容，计算每个文件属于每个类别的概率。最终输出 <文件名, <类名, 概率>>。

**Reducer 阶段：**Reducer 会根据输入的多个类别及其对应的概率，选择概率最大的一类作为文件的预测类别。

该部分主要由 PredictMapper 类、PredictReducer 类、Prediction 类实现，下面分别介绍各个部分。

## 1. PredictMapper 类

PredictMapper 是 Map 阶段的实现类。它的作用是接收每个文件的内容，计算每个类别的概率，然后将文件名与类别及其概率一起输出。

输入数据为<文件名, 类名@单词 1,单词 2,...>, 输出数据为<文件名, <类名, 概率>>。函数的流程如下：

①从输入中解析文件内容，并提取出文件的单词列表。

②遍历所有的类别（如 AUSTR、BRAZ、INDIA），计算文件属于每个类别的概率。贝叶斯分类公式为：

$$P(\text{类名} | w_1, w_2, \dots, w_n) = P(\text{类名}) \times \prod_{i=1}^n P(w_i | \text{类名})$$

其中  $P(\text{类名})$  是先验概率（由训练数据计算得到）， $P(w_i | \text{类名})$  是每个单词在该类别下的条件概率（也是由训练数据计算得到）。这两个概率值都会存储在 priors 和 termProbability 中。

③对于每个单词，如果它不在训练数据的词汇集中，会使用平滑处理，计算一个小的概率值。

④将每个类别的概率结果与文件名一起输出。

```
public static class PredictMapper extends Mapper<Text, Text, Text, Text> {
    @Override
    protected void map(Text key, Text value, Context context) throws IOException,
        InterruptedException {
        String[] parts = value.toString().split("@");
        String[] split = parts[1].split(",");           // 提取单词列表
        for (String className : Config.CLASS_ARRAY) {
            String[] split2 = split[1].split(",");       //当前单词组
```

```
double prior = priors.get(className);    //先验概率
for (String s : split2) {

    // 求文档属于该类的概率: 公式 = 先验概率 + 每个单词的概率
    double temp = 0.0;
    if (!termSet.contains(s)) {
        temp=Math.log(1/(classTermSum.get(className)+
termSet.size()));
    } else {
        temp = termProbability.get(className + "," + s);
    }
    prior += temp;
}
String v = className + "," + prior;
context.write(key, new Text(v));
}
}
```

## 2. PredictReducer 类

PredictReducer 是 Reduce 阶段的实现类。它的作用是接收来自 Mapper 的每个文件的各类概率，选择最大概率的类别作为文件的预测类别。

输入数据为<文件名, {<类名 1, 概率 1>, <类名 2, 概率 2>, ...}>, 输出数据为<文件名, 类名>, 其中类名是预测的类别。具体流程如下:

①对于每个文件（以 key 为文件名），遍历所有类别的概率（由 PredictMapper 输出）。

②选择最大概率的类别作为文件的预测类别。

③将预测类别写入上下文，作为最终输出。

```
public static class PredictReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
        String maxClass = "";
        double maxPro = Double.NEGATIVE_INFINITY;

        // 遍历所有类别的概率，选择最大概率的类别
        for (Text value : values) {
            String className = value.toString().split(",")[0];
            double pro = Double.valueOf(value.toString().split(",")[1]);
            if (pro > maxPro) {
                maxClass = className; maxPro = pro;
            }
        }
    }
}
```

```
context.write(key, new Text(maxClass)); // 输出预测结果
    }
}
```

### 3. Prediction 类

Predition 类负责计算先验概率、条件概率和训练数据的词汇集合。其中有 3 个方法，作用如下：

输入数据为<文件名, {<类名 1, 概率 1>, <类名 2, 概率 2>, ...}>, 输出数据为<文件名, 类名>, 其中类名是预测的类别。具体流程如下：

①calPriors 方法：计算每个类别的先验概率 $P(\text{类名})$ ，它依赖于训练数据中的各个类的频率。

```
private static void calPriors() throws IOException {
    double totalSum = 0.0;
    List<String> classNames = new ArrayList<>();
    List<Integer> classCounts = new ArrayList<>();

    // 读取类的统计信息
    try (FileSystem fs = FileSystem.get(URI.create(Config.CLASS_SUM_LOC), new
Configuration());
        InputStream in = fs.open(new Path(Config.CLASS_SUM_LOC));
        Scanner scanner = new Scanner(in)) {

        while (scanner.hasNext()) {
            String className = scanner.next();
            int classCount = scanner.nextInt();
            classNames.add(className);
            classCounts.add(classCount);
            totalSum += classCount;
        }
    }

    // 计算每个类的先验概率
    for (int i = 0; i < classNames.size(); i++) {
        priors.put(classNames.get(i), Math.log(classCounts.get(i) / totalSum));
    }
}
```

②calTermProbability 方法：计算每个单词在各个类别下的条件概率  $P(\text{单词} | \text{类名})$ 。

```
private static void calTermProbability() throws IOException {
    Map<String, Integer> termCounts = new HashMap<>();
    double austrSum = 0.0, brazSum = 0.0, indiaSum = 0.0;

    // 读取训练集中的单词统计信息
```

```
try (FileSystem fs = FileSystem.get(URI.create(Config.TERM_LOC),
new Configuration());InputStream in = fs.open(new Path(Config.TERM_LOC));
Scanner scanner = new Scanner(in)) {

    while (scanner.hasNext()) {
        String termName = scanner.next();
        int termCount = scanner.nextInt();
        String[] split = termName.split(",");
        switch (split[0]) {
            case Config.AUSTR: austrSum += termCount; break;
            case Config.BRAZ: brazSum += termCount; break;
            case Config.INDIA: indiaSum += termCount; break;
        }

        termSet.add(split[1]); // 添加到单词集合
        termCounts.put(termName, termCount);
    }

    // 计算每个类别的单词总数
    classTermSum.put(Config.AUSTR, austrSum);
    classTermSum.put(Config.BRAZ, brazSum);
    classTermSum.put(Config.INDIA, indiaSum);

    // 计算条件概率
    int vocabularySize = termSet.size();
    for (String word : termSet) {
        calculateWordProbability(Config.AUSTR, word, austrSum, termCounts,
vocabularySize);
        calculateWordProbability(Config.BRAZ, word, brazSum, termCounts,
vocabularySize);
        calculateWordProbability(Config.INDIA, word, indiaSum, termCounts,
vocabularySize);
    }
}
```

③calculateWordProbability 方法：为每个单词计算条件概率，并处理平滑问题。

```
private static void calculateWordProbability(String className, String word, double
classSum,Map<String, Integer> termCounts, int vocabSize) {
    String key = className + "," + word;
    int wordCount = termCounts.getOrDefault(key, 0);
    double probability = Math.log((wordCount + 1) / (classSum + vocabSize));
    termProbability.put(key, probability);
}
```

## 4 实验与结果分析

### 4.1 实验环境

实验开发环境为 Linux 操作系统，使用 OpenJDK 11.0.25、Apache Maven 3.6.3 和 Hadoop 2.10.2 本地模式，开发工具为 Visual Studio Code 1.96.3，硬件配置为 64 核 Intel Xeon Gold 6326 CPU。具体配置如表 4-1 所示。

表 4-1 实验环境配置

配置项	配置信息
操作系统	Linux version 5.15.0-124-generic
CPU	64 x Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz
编程语言	Java
JDK 版本	Openjdk 11.0.25
开发工具	Visual Studio Code 1.96.3
构建工具	Apache Maven 3.6.3
Hadoop 版本	Hadoop 2.10.2
Hadoop 集群模式	本地模式 (Local)

配置好的 Hadoop 集群环境如图 4-1 所示。

```

● xjh@host2:~/mapreduce/project$ jps
3339939 DataNode
3339728 NameNode
3340433 SecondaryNameNode
1608700 Jps
1592926 org.eclipse.equinox.launcher_1.6.900.v20240613-2009.jar
1819537 NodeManager
1818582 ResourceManager
    
```

图 4-1 Hadoop 集群环境

### 4.2 实验数据集

为了实现更好地分类效果，我们选择了 NBCorpus\Country 中数量较多的 3 个目录作为数据集，并将其按照 7:3 的比例划分为训练集和测试集，打乱文件原始顺序，确保每类数据具有代表性并能有效进行分类。具体目录如所表 4-2 示。

表 4-2 数据集分布

类别	训练集	测试集	总数
AUSTR	213	92	305
BRAZ	140	60	200
INDIA	228	98	326

对数据集的打乱和划分由 Python 文件 tool.py 实现,运行结果如图 4-3 所示。

```
xjh@host2:~/mapreduce$ python3 tool.py
正在处理目录: NBCorpus/Country/AUSTR
总共有 305 个文件, 训练集包含 213 个文件, 测试集包含 92 个文件。
正在处理目录: NBCorpus/Country/BRAZ
总共有 200 个文件, 训练集包含 140 个文件, 测试集包含 60 个文件。
正在处理目录: NBCorpus/Country/INDIA
总共有 326 个文件, 训练集包含 228 个文件, 测试集包含 98 个文件。
```

图 4-2 划分数据集

接着将数据集上传到 Hadoop 分布式文件系统上,使用以下命令:

```
hadoop fs -put src/resources/Country/Test hdfs://host2:9000/CountryTest
hadoop fs -put src/resources/Country/Train hdfs://host2:9000/CountryTrain
```

运行截图及上传成功后 HDFS 的目录分别如图 4-4 和图 4-4 所示。

```
xjh@host2:~/mapreduce/project$ hadoop fs -put src/resources/Country/Test hdfs://host2:9000/CountryTest
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-2.10.2.jar) to method sun.security.krb5.Config.getInstance()
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
25/01/14 07:49:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

图 4-3 上传数据集

## Browse Directory

/CountryTest

Go!

Show

25

entries

Search:

<input type="checkbox"/>	<div><div></div></div> Permission	<div><div></div></div> Owner	<div><div></div></div> Group	<div><div></div></div> Size	<div><div></div></div> Last Modified	<div><div></div></div> Replication	<div><div></div></div> Block Size	<div><div></div></div> Name	<div><div></div></div>
<input type="checkbox"/>	<a href="#">drwxr-xr-x</a>	<a href="#">xjh</a>	<a href="#">supergroup</a>	0 B	Jan 14 15:49	<a href="#">0</a>	0 B	<a href="#">AUSTR</a>	<div><div></div></div>
<input type="checkbox"/>	<a href="#">drwxr-xr-x</a>	<a href="#">xjh</a>	<a href="#">supergroup</a>	0 B	Jan 14 15:49	<a href="#">0</a>	0 B	<a href="#">BRAZ</a>	<div><div></div></div>
<input type="checkbox"/>	<a href="#">drwxr-xr-x</a>	<a href="#">xjh</a>	<a href="#">supergroup</a>	0 B	Jan 14 15:49	<a href="#">0</a>	0 B	<a href="#">INDIA</a>	<div><div></div></div>

Showing 1 to 3 of 3 entries

Previous

1

Next

图 4-4 Web 页面输入文档

## 4.3 实验运行分析

### 4.3.1 Job1: ClassCount

在运行第一个任务时，共启动了 3 个 Map 任务和 1 个 Reduce 任务，运行截图如图 4-5 和图 4-6 所示。Map 任务的数量是根据输入数据的文件数量和分片数决定的，日志中显示总共有 3 个分片，因此启动了 3 个 Map 任务。这些 Map 任务分别处理输入文件中的数据，并为每个类统计文件数量。在运行过程中，Map 任务的总执行时间较长，表明数据处理的计算量较大。

```
xjh@host2:~/mapreduce/project$ hadoop jar target/xjh.jar com.xjh.jobs.ClassCount
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-2.10.2.jar) to method sun.security.krb5.Config.getInstance()
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
25/01/14 08:13:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/01/14 08:13:31 INFO client.RMProxy: Connecting to ResourceManager at host2/127.0.1.1:8032
25/01/14 08:13:31 WARN ipc.Client: Address change detected. Old: host2/10.10.10.2:9000 New: host2/127.0.1.1:9000
25/01/14 08:13:31 WARN ipc.Client: Exception when handle ConnectionFailure: Connection refused
25/01/14 08:13:31 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
25/01/14 08:13:31 INFO input.FileInputFormat: Total input files to process : 3
25/01/14 08:13:31 INFO mapreduce.JobSubmitter: number of splits:3
25/01/14 08:13:31 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1736345261028_0010
25/01/14 08:13:31 INFO conf.Configuration: resource-types.xml not found
25/01/14 08:13:31 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/01/14 08:13:31 INFO resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
25/01/14 08:13:31 INFO resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
25/01/14 08:13:31 INFO impl.YarnClientImpl: Submitted application application_1736345261028_0010
```

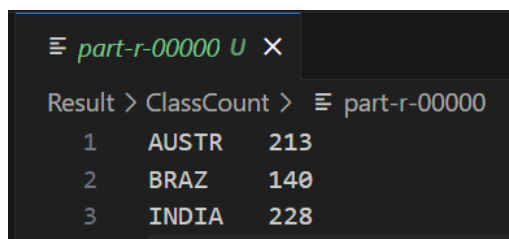
图 4-5 ClassCount 运行截图 1

```
Job Counters
  Killed map tasks=1
  Launched map tasks=3
  Launched reduce tasks=1
  Other local map tasks=3
  Total time spent by all maps in occupied slots (ms)=5966
  Total time spent by all reduces in occupied slots (ms)=2049
  Total time spent by all map tasks (ms)=5966
  Total time spent by all reduce tasks (ms)=2049
  Total vcore-milliseconds taken by all map tasks=5966
  Total vcore-milliseconds taken by all reduce tasks=2049
  Total megabyte-milliseconds taken by all map tasks=6109184
  Total megabyte-milliseconds taken by all reduce tasks=2098176
```

图 4-6 ClassCount 运行截图 2

ClassCount 运行结果如图 4-7 所示，可以正确统计出每个类别的文件数量。同时在 web 端也有相应的结果目录。

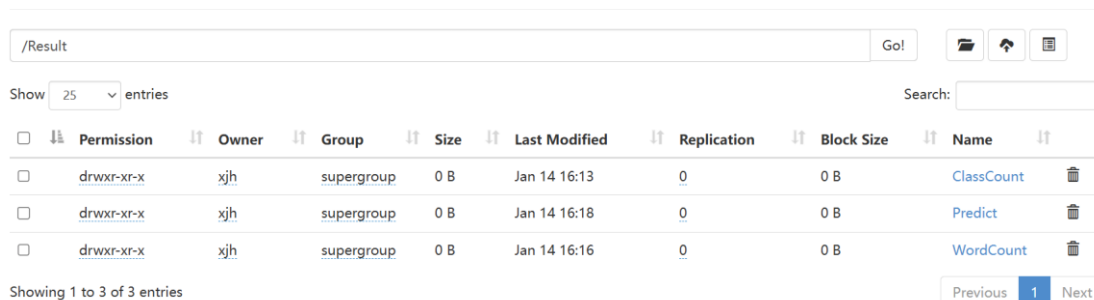




	Country	Count
1	AUSTR	213
2	BRAZ	140
3	INDIA	228

图 4-7 ClassCount 结果

## Browse Directory



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	xjh	supergroup	0 B	Jan 14 16:13	0	0 B	ClassCount
drwxr-xr-x	xjh	supergroup	0 B	Jan 14 16:18	0	0 B	Predict
drwxr-xr-x	xjh	supergroup	0 B	Jan 14 16:16	0	0 B	WordCount

图 4-8 Web 页面输出文档

### 4.3.2 Job2: WordCount

WordCount 的运行截图如图 4-9 和图 4-10 所示。在实验过程中，执行了 3 个 Map 任务和 1 个 Reduce 任务，输出结果中有多个不同频次的单词项。Map 任务总共消耗了 5663 毫秒，Reduce 任务消耗了 1689 毫秒。

```
xjh@host2:~/mapreduce/project$ hadoop jar target/xjh.jar com.xjh.jobs.WordCount
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/usr/local/hadoop/share/hadoop/common/lib/hadoop-auth-2.10.2.jar) to method sun.security.krb5.Config.getInstance()
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
25/01/14 08:16:44 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
25/01/14 08:16:44 INFO client.RMProxy: Connecting to ResourceManager at host2/127.0.1.1:8032
25/01/14 08:16:44 WARN ipc.Client: Address change detected. Old: host2/10.10.10.2:9000 New: host2/127.0.1.1:9000
25/01/14 08:16:44 WARN ipc.Client: Exception when handle ConnectionFailure: Connection refused
25/01/14 08:16:45 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
25/01/14 08:16:45 INFO input.FileInputFormat: Total input files to process : 3
25/01/14 08:16:45 INFO mapreduce.JobSubmitter: number of splits:3
25/01/14 08:16:45 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1736345261028_0011
25/01/14 08:16:45 INFO conf.Configuration: resource-types.xml not found
```

图 4-9 WordCount 运行截图 1

输出结果文件的一部分如图 4-11 所示。结果表明，每个类别中每个单词的频次被成功统计，并且按照 <类别, 单词> 的形式输出。通过这些统计结果，可以进一步分析每个类别中的单词分布，有助于在后续的分类任务中计算每个单词



的条件概率，并为分类器提供特征数据。

```
Job Counters
  Killed map tasks=1
  Launched map tasks=3
  Launched reduce tasks=1
  Other local map tasks=3
  Total time spent by all maps in occupied slots (ms)=5663
  Total time spent by all reduces in occupied slots (ms)=1689
  Total time spent by all map tasks (ms)=5663
  Total time spent by all reduce tasks (ms)=1689
  Total vcore-milliseconds taken by all map tasks=5663
  Total vcore-milliseconds taken by all reduce tasks=1689
  Total megabyte-milliseconds taken by all map tasks=5798912
  Total megabyte-milliseconds taken by all reduce tasks=1729536
```

图 4-10 WordCount 运行截图 2

```
≡ part-r-00000 U ×
Result > WordCount > ≡ part-r-00000
1  AUSTR,0 6
2  AUSTR,0.0 7
3  AUSTR,0.007 1
4  AUSTR,0.01 3
5  AUSTR,0.01goodman 1
```

图 4-11 WordCount 运行结果

### 4.3.3 Job3: Predict

在第三个任务的实验中，执行了 3 个 Map 任务和 1 个 Reduce 任务。由于输入数据集包含了 3 个文件，所以 Hadoop 自动将每个文件分配给一个独立的 Map 任务。因此，共启动了 3 个 Map 任务，并最终产生了 1 个 Reduce 任务来进行结果的汇总和输出。结果如图 4-12 和图 4-13 所示。

输出的文件列表中，每一行都包含了一个文件名及其对应的预测类别，格式为 <文件名> <类别>，结果如图 4-14 所示。

```

xjh@host2:~/mapreduce/project$ hadoop jar target/xjh.jar com.xjh.jobs.Predict
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil (file:/usr/local/hadoop/s
hare/hadoop/common/lib/hadoop-auth-2.10.2.jar) to method sun.security.krb5.Config.getInstance()
WARNING: Please consider reporting this to the maintainers of org.apache.hadoop.security.authentication.util.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
25/01/15 02:33:06 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
25/01/15 02:33:06 INFO client.RMProxy: Connecting to ResourceManager at host2/127.0.1.1:8032
25/01/15 02:33:07 WARN ipc.Client: Address change detected. Old: host2/10.10.10.2:9000 New: host2/127.0.1.1:9000
25/01/15 02:33:07 WARN ipc.Client: Exception when handle ConnectionFailure: Connection refused
25/01/15 02:33:07 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool
interface and execute your application with ToolRunner to remedy this.
25/01/15 02:33:07 INFO input.FileInputFormat: Total input files to process : 3
25/01/15 02:33:07 INFO mapreduce.JobSubmitter: number of splits:3
25/01/15 02:33:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1736345261028_0013
25/01/15 02:33:08 INFO conf.Configuration: resource-types.xml not found
25/01/15 02:33:08 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.

```

图 4-12 Predict 运行截图 1

```

Job Counters
  Launched map tasks=3
  Launched reduce tasks=1
  Other local map tasks=3
  Total time spent by all maps in occupied slots (ms)=7932
  Total time spent by all reduces in occupied slots (ms)=2147
  Total time spent by all map tasks (ms)=7932
  Total time spent by all reduce tasks (ms)=2147
  Total vcore-milliseconds taken by all map tasks=7932
  Total vcore-milliseconds taken by all reduce tasks=2147
  Total megabyte-milliseconds taken by all map tasks=8122368
  Total megabyte-milliseconds taken by all reduce tasks=2198528

```

图 4-13 Predict 运行截图 2

part-r-00000 U X		
Result > Predict > part-r-00000		
1	477935newsML.txt	AUSTR
2	478258newsML.txt	BRAZ
3	478299newsML.txt	AUSTR
4	478731newsML.txt	BRAZ
5	478745newsML.txt	BRAZ
6	478747newsML.txt	BRAZ

图 4-14 Predict 运行结果

## 4.4 实验结果分析

我们通过 Python 脚本对实验结果进行分析，并计算了各项指标，运行结果如图 4-15 所示。其中各项指标计算公式如下：

TP：真正例，预测正确为某类别的文件数。

FN：假负例，真实类别为某类别，但预测为其他类别的文件数。

FP：假正例，预测为某类别，真实类别为其他类别的文件数。

TN：真负例，既不属于某类别也未被预测为该类别的文件数。

精确度：  $\text{Precision} = \frac{TP}{TP+FP}$

召回率：  $\text{Recall} = \frac{TP}{TP+FN}$

F1 值：  $\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$

Class	TP	FN	FP	TN	Precision	Recall	F1
AUSTR	91	1	4	154	0.9579	0.9891	0.9733
BRAZ	59	1	1	189	0.9833	0.9833	0.9833
INDIA	95	3	0	152	1.0000	0.9694	0.9845
Macro Average	-	-	-	-	0.9804	0.9806	0.9804

图 4-15 Python 脚本运行

实验结果如表 4-3 所示，宏平均 Precision 为 0.9804，宏平均 Recall 为 0.9806，宏平均 F1 为 0.9804。模型在分类所有三个类别（AUSTR、BRAZ、INDIA）时的表现良好。特别是在 INDIA 类别上，精确度达到了完美的 1.0000，说明该类的分类非常准确。其他类别的精确度和召回率也都表现优秀，尤其是 BRAZ 类的精确度和召回率都达到了较高的值。

表 4-3 实验结果

类别	TP	FN	FP	TN	Precision	Recall	F1
AUSTR	91	1	4	154	0.9579	0.9891	0.9733
BRAZ	59	1	1	189	0.9833	0.9833	0.9833
INDIA	95	3	0	152	1.0000	0.9694	0.9845
Macro Average	-	-	-	-	0.9804	0.9806	0.9804