实验二

211275009 陈铭浩

1.背景

在借贷交易中，银行和其他金融机构通常提供资金给借款人，期望借款人能够按时还款本金和利
息。然而，由于各种原因，有时借款人可能无法按照合同规定的方式履行还款义务，从而导致贷
款违约。本次实验以银行贷款违约为背景，选取了约30万条贷款信息，包含在application_data.csv文件中，数据描述包含在columns_description.csv文件夹中。
数据来源：https://www.kaggle.com/datasets/mishra5001/credit-card/data

2.任务一

2.1.任务描述

编写MapReduce程序，统计数据集中违约和非违约的数量，按照标签TARGET进行输出，即1代
表有违约的情况出现，0代表其他情况。
输出格式：
<标签><交易数量>
例：
1 100

2.2.设计思路

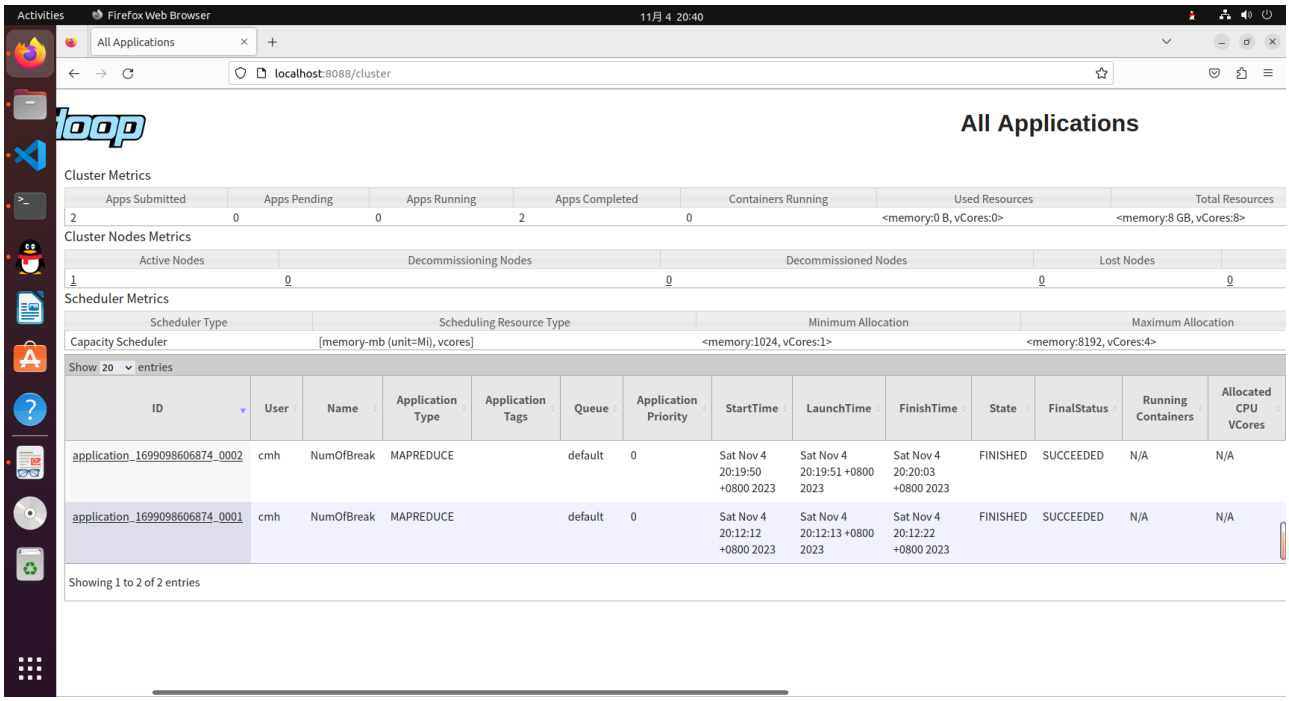
任务一可看作是wordcount任务的一个简单变体，考虑在mapper中直接取出每一行数据（除第一行标题行外）的target标签并计数，在reducer中对每一种target标签（0/1）计数即可。

```java
public static class NumOfBreakMapper extends Mapper<LongWritable,Text,Text,IntWritable>{
    private final static IntWritable one =new IntWritable(value:1);
    protected void map(LongWritable key,Text value,Context context) throws IOException,InterruptedException{
        String line = value.toString();
        String[] fields = line.split(",");
        if (key.get()!=0){
            String target=fields[fields.length-1].trim();
            context.write(new Text(target), one);
        }
    }
}
```

```java
public static class NumOfBreakReducer extends Reducer<Text,IntWritable,Text,IntWritable>{
    protected void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,InterruptedException{
        int sum=0;
        for (IntWritable val:values){
            sum +=val.get();
        }
        context.write(key,new IntWritable(sum));
    }
}
```

```java
public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, jobName:"NumOfBreak");

    job.setJarByClass(cls:NumOfBreak.class);
    job.setMapperClass(cls:NumOfBreakMapper.class);
    job.setReducerClass(cls:NumOfBreakReducer.class);

    job.setOutputKeyClass(theClass:Text.class);
    job.setOutputValueClass(theClass:IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(verbose:true) ? 0 : 1);
}
```

2.3.运行结果




```
cmh@cmh-virtual-machine:~/hadoop/hadoop-3.3.6/experiment2$ hdfs dfs -get output
output
cmh@cmh-virtual-machine:~/hadoop/hadoop-3.3.6/experiment2$ cat output/*
0       282686
1       24825
```

3.任务二

3.1.任务描述

编写MapReduce程序，统计一周当中每天申请贷款的交易数
WEEKDAY_APPR_PROCESS_START，并按照交易数从大到小进行排序。
输出格式：
<交易数量>
例：
Sunday 16000

3.2.设计思路

首先在mapper中直接取出每一行数据（除第一行标题行外）的WEEKDAY_APPR_PROCESS_START标签并计数，在reducer中对每一种Weekday累计计数。为了按照交易数对结果从大到小排序，我使用了treemap，并利用Collections.reverseOrder()将元素逆序排列，最后在cleanup中输出排序好的结果

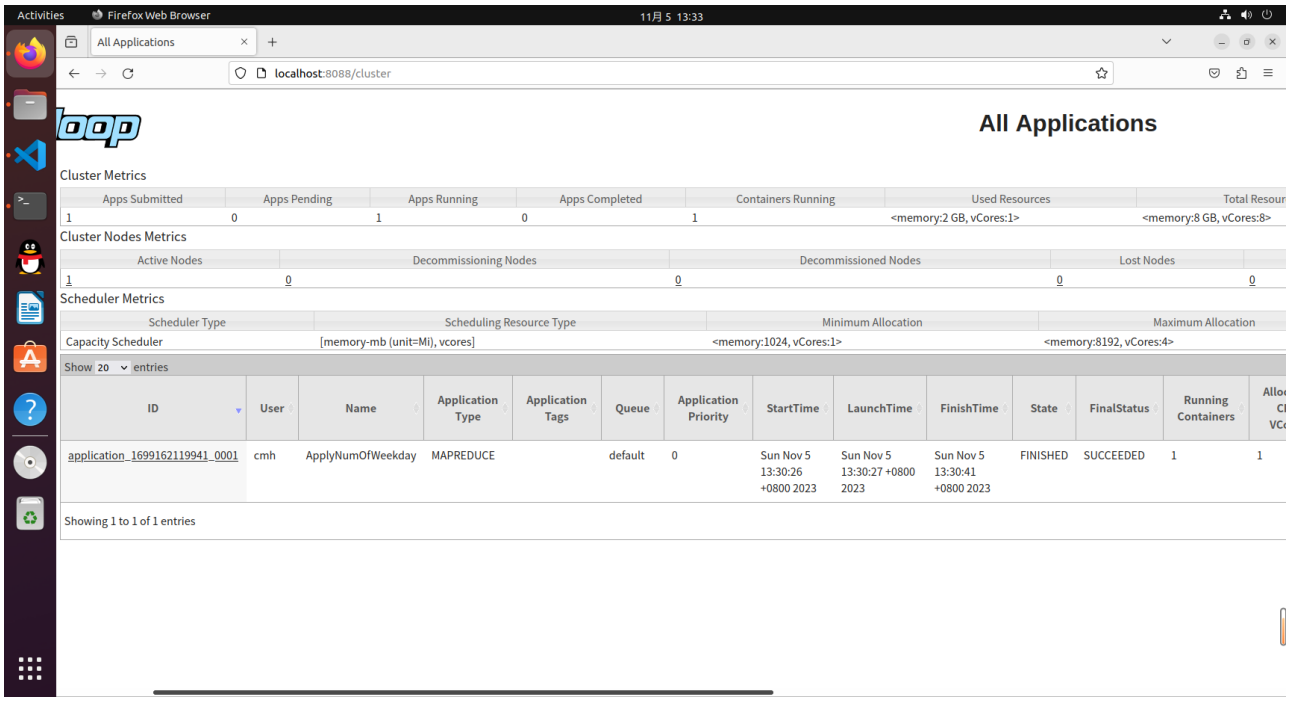
> *TreeMap是一种基于红黑树实现的有序映射数据结构，它根据键的自然顺序或自定义排序顺序来维护键-值对的有序性。默认情况下，TreeMap会按照键的自然顺序（升序）来排序。如果想要逆序（降序）排序，可以使用Collections.reverseOrder()来创建一个比较器，它会将元素逆序排列。*

> *在Hadoop MapReduce中，cleanup函数是一个用于Mapper和Reducer任务的生命周期方法之一。cleanup函数在Map或Reduce任务执行结束后被调用，用于执行一些清理工作。具体而言：*
> *对于Mapper任务：cleanup函数会在Mapper任务执行完毕后调用。你可以在cleanup函数中进行一些资源释放、缓存刷新等清理工作；*
> *对于Reducer任务：cleanup函数会在Reducer任务执行完毕后调用。它可以用于执行一些清理操作，例如将数据写入数据库、关闭文件句柄等。*
> *cleanup函数通常用于处理与MapReduce任务相关的资源管理，以确保任务执行后资源被正确释放，或者用于执行一些最终的计算和输出操作。*

```java
public class ApplyNumOfWeekday {
    public static class ApplyNumOfWeekdayMapper extends Mapper<LongWritable,Text,Text,IntWritable>{
        private final static IntWritable one =new IntWritable(value:1);
        protected void map(LongWritable key,Text value,Context context) throws IOException,InterruptedException{
            String line = value.toString();
            String[] fields = line.split(",");
            if (key.get()!=0){
                String weekday=fields[25].trim();
                context.write(new Text(weekday), one);
            }
        }
    }

    public static class ApplyNumOfWeekdayReducer extends Reducer<Text,IntWritable,Text,IntWritable>{
        TreeMap<Integer,String> sortedWeekdays = new TreeMap<>(Collections.reverseOrder());
        protected void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,InterruptedException{
            int sum=0;
            for (IntWritable val:values){
                sum +=val.get();
            }
            sortedWeekdays.put(sum,key.toString());
            //context.write(key,new IntWritable(sum));
        }
        protected void cleanup(Context context) throws IOException,InterruptedException{
            for (Integer count : sortedWeekdays.keySet()) {
                context.write(new Text(sortedWeekdays.get(count)), new IntWritable(count));
            }
        }
    }
```

```java
public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, jobName:"ApplyNumOfWeekday");

    job.setJarByClass(cls:ApplyNumOfWeekday.class);
    job.setMapperClass(cls:ApplyNumOfWeekdayMapper.class);
    job.setCombinerClass(cls:ApplyNumOfWeekdayReducer.class);
    job.setReducerClass(cls:ApplyNumOfWeekdayReducer.class);

    job.setOutputKeyClass(theClass:Text.class);
    job.setOutputValueClass(theClass:IntWritable.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    System.exit(job.waitForCompletion(verbose:true) ? 0 : 1);
}
```

3.3.运行结果




```
cmh@cmh-virtual-machine:~/hadoop/hadoop-3.3.6/experiment2$ cat output/*
TUESDAY 53901
WEDNESDAY       51934
MONDAY  50714
THURSDAY        50591
FRIDAY  50338
SATURDAY        33852
SUNDAY  16181
```

4.任务三

4.1.任务描述

根据application_data.csv中的数据，基于MapReduce建立贷款违约检测模型，并评估实验结果的
准确率。
说明：
1、该任务可视为一个“二分类”任务，因为数据集只存在两种情况，违约（Class=1）和其他（Class=0）。
2、可根据时间特征的先后顺序按照8：2的比例将数据集application_data.csv拆分成训练集和测
试集，时间小的为训练集，其余为测试集；也可以按照8：2的比例随机拆分数据集。最后评估模
型的性能，评估指标可以为accuracy、f1-score等。
3、基于数据集application_data.csv，可以自由选择特征属性的组合，自行选用分类算法对目标
属性TARGET进行预测。

4.2.数据预处理

详见data_clean_divide.ipynb文件

- 缺失值处理
- 唯一值
- 连续特征归一化
- 正负样本不均衡，采用下采样策略
- 将数据随机划分为训练集和测试集（4：1）

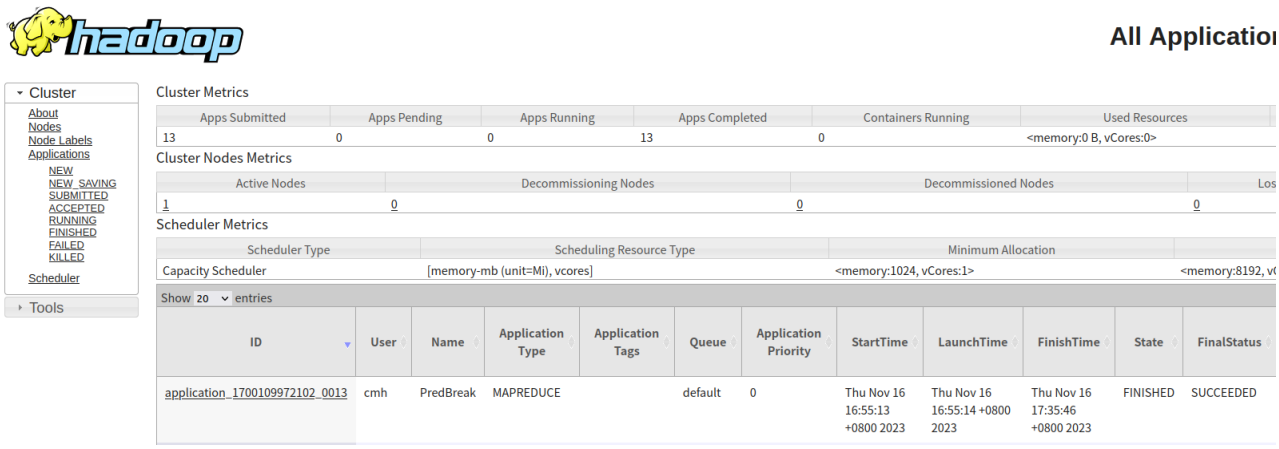
4.3.设计思路

模型：KNN分类并行化算法
基本思路：将测试样本数据分块后分布在不同的节点上进行处理，将训练样本数据文件放在DistributedCache中供每个节点共享访问。
map阶段：针对每个读出的测试样本，与每一个训练样本计算距离（这里考虑到特征中既有

连续型又有离散型，使用欧式距离与汉明距离加权平均的混合距离），找出距离最小的k个训练样本，建立一个带加权的投票表决计算模型，从而计算出测试样本的分类标记预测值。reduce阶段：分别统计TP、FP、TN、FN数目，并在cleanup中计算accuracy、precision、recall和f1-score等指标，输出结果。

ps:具体代码见github仓库

4.4.运行结果




```
Accuracy:        0.5688785800726099
Precision:       0.5658545869813475
Recall: 0.5985504328568553
F1-Score:        0.5817434693278545
```

4.5.反思

- KNN算法的优点是简单，易于理解，且无需估计参数；但同时，它的缺点也非常明显，它是懒惰算法，对测试样本分类时的计算量大，内存开销也大，数据量较大时很容易造成程序运行时间过长，而且KNN必须指定K值，如果K值选择不当，则最后的分类精度无法保证。
- 改进方向：

1. 交叉验证选取合适的k值
2. 混合距离计算中欧氏距离与汉明距离权重的分配
3. 其它分类算法（贝叶斯/决策树/...）