实验二

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

任务

任务一

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

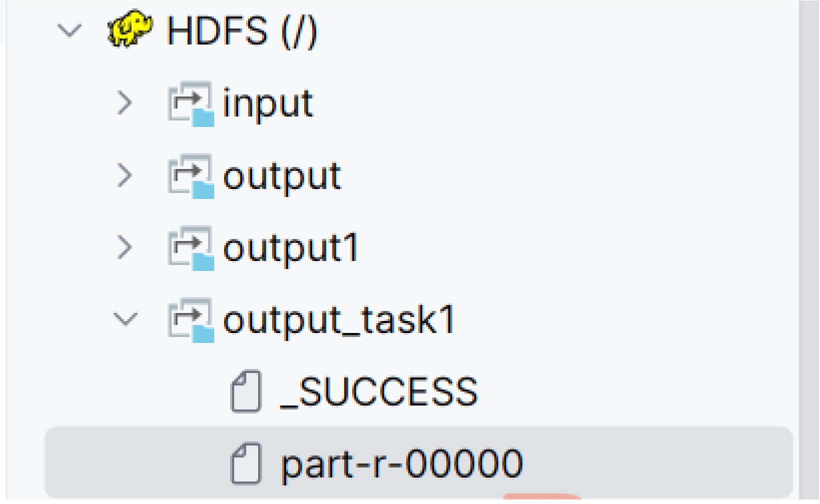
代码:

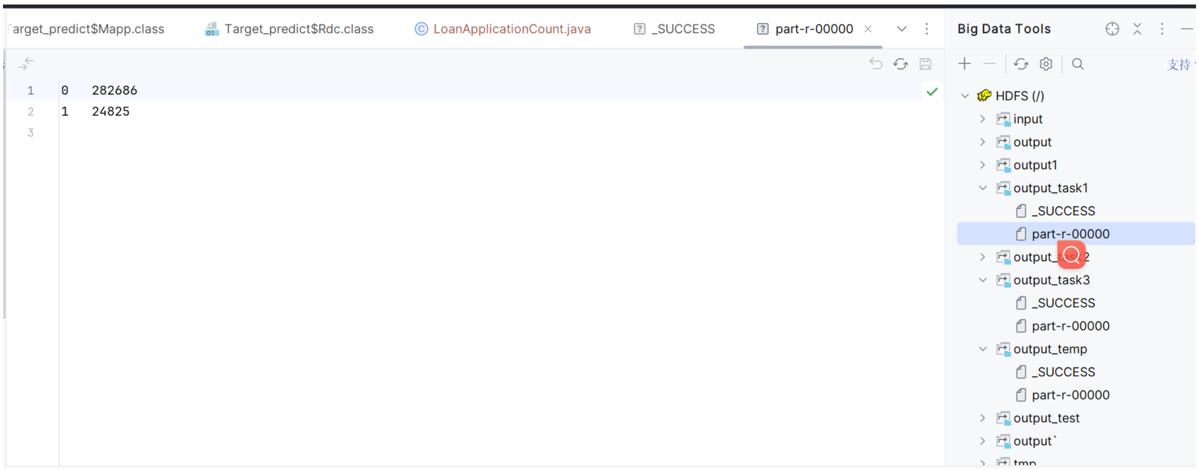
```
package com.TargetCountMapper;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class LoanApplicationCount {

    public static class LoanApplicationMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            if(key.get()==0){
                return;
            }
            String[] parts = value.toString().split(",");
            word.set(parts[parts.length - 1]);//最后一项为是否违约
            context.write(word, one);
        }
    }

    public static class LoanApplicationReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable Num = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

            int count=0;
            for (IntWritable val : values) {
                count+=val.get();
            }
            Num.set(count);
            context.write(key, Num);
        }
    }
    public static void main(String[] args) throws
Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "loan
application count");
        job.setJarByClass(LoanApplicationCount.class);
        job.setMapperClass(LoanApplicationMapper.class);

job.setCombinerClass(LoanApplicationReducer.class);

job.setReducerClass(LoanApplicationReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 :
1);
    }
}
```

运行和用途

它的主要功能是统计Target在数据集中出现的次数，与之前的WORD_COUNT代码原理相同，较为简单。

运行结果





```
0	282686
1	24825
```

- 由于我是使用的IDEA远程连接，所以没有从虚拟机中输出运行成功界面！
- 根据统计量得到了282686次0,以及24825次1。

任务二

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>

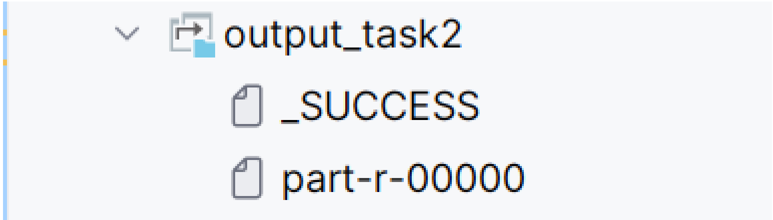
例： Sunday 16000

```
package com.TargetCountMapper;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.io.IOException;

public class WeekdayLoanCount {

    public static class LoanApplicationMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            if(key.get()==0){
                return;
            }
            String[] parts = value.toString().split(",");
            word.set(parts[25]);//使用星期几为键值
            context.write(word, one);
        }
    }

    public static class LoanApplicationReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private Map<Text, Integer> counts = new HashMap<>();
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            int count=0;

            for (IntWritable val : values) {
                count+=val.get();
            }
            counts.put(new Text(key), count);
        }
        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
            // 使用 TreeMap 对 counts 进行排序
            TreeMap<Text, Integer> sortedCounts = new TreeMap<>(counts);

            for (Map.Entry<Text, Integer> entry : sortedCounts.entrySet()) {
                result.set(entry.getValue());
                context.write(entry.getKey(), result);
            }
        }
    }

    public static void main(String[] args) throws
Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "weekday loan
count");
        job.setJarByClass(LoanApplicationCount.class);
        job.setMapperClass(LoanApplicationMapper.class);

job.setCombinerClass(LoanApplicationReducer.class);

job.setReducerClass(LoanApplicationReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 :
1);
    }
}
```

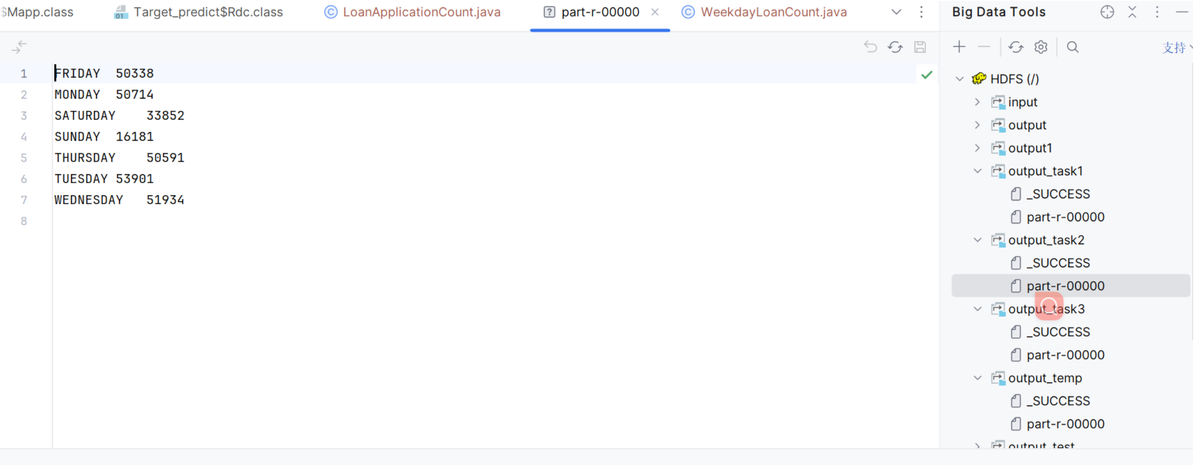
运行和用途

它的主要功能是统计Target在星期几的分布情况，其主要原理仍与WORD_COUNT相同，通过MAP进行单词的读数，通过reduce进行频数的统计最后输出结果

运行结果



output_task2
_SUCCESS
part-r-00000



```
FRIDAY	50338
MONDAY	50714
SATURDAY	33852
SUNDAY	16181
THURSDAY	50591
TUESDAY	53901
WEDNESDAY	51934
```

任务三

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。

预处理

```
package com.TargetCountMapper;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```java
import java.io.IOException;

public class LoanDefaultPredict {

    public static class LoanApplicationMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            if(key.get()==0){
                return;
            }
            String[] parts = value.toString().split(",");
            for(int i = 0; i < parts.length-1; i++){
                if(parts[parts.length-1].equals("0")){
                    word.set(i+" "+"0" +" "+parts[i]);
                    context.write(word, one);
                }
                else{
                    word.set(i+" "+"1" +" "+parts[i]);
                    context.write(word, one);
                }
            }

        }
    }

    public static class LoanApplicationReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable Num = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

            int count=0;
            for (IntWritable val : values) {
                count+=val.get();
            }
            Num.set(count);
            context.write(key, Num);
        }
    }

    public static void main(String[] args) throws
Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "loan
application count");
        job.setJarByClass(LoanApplicationCount.class);
        job.setMapperClass(LoanApplicationMapper.class);

job.setCombinerClass(LoanApplicationReducer.class);

job.setReducerClass(LoanApplicationReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new
Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 :
1);
    }
}
```

- 首先通过mapreduce进行一次预处理，依次取列元素与Target构成String字符串，进行输出，便于后续的操作

预测

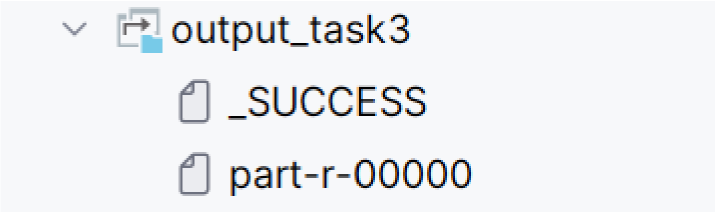
```
package com.TargetCountMapper;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.HashMap;
import java.util.Map;

public class FinalPredict{


    public static class LoanApplicationMapper extends
Mapper<LongWritable, Text, Text, IntWritable> {
        private Map<String, String> myMap = new
HashMap<>();

        @Override
        protected void setup(Context context) throws
IOException, InterruptedException {
            try {
                Path[] cacheFiles =
DistributedCache.getLocalCacheFiles(context.getConfigura
tion());
                if (cacheFiles != null &&
cacheFiles.length > 0) {
                    String line;
                    BufferedReader reader = new
BufferedReader(new
FileReader(cacheFiles[0].toString()));
                    while ((line = reader.readLine())
!= null) {
                        String[] parts =
line.split("\\s+");
                        String key = parts[0]+"
"+parts[1]+" "+parts[2];
                        myMap.put(key,parts[3]);
                    }
                    reader.close();
                }
            } catch (IOException e) {
                System.err.println("Exception reading
DistributedCache: " + e);
            }
        }
        private final static IntWritable one = new
IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value,
Context context) throws IOException,
InterruptedException {
            //读取测试集数据
            double zero_F = 11.36;
            double one_F = 1;
            Text target = new Text();

            if(key.get()==0){
                return;
            }
            String[] parts =
value.toString().split(",");
            for (int i = 0; i < parts.length - 1; i++)
{
                String xnj = Integer.toString(i);
                String classLabel = parts[parts.length
- 1];
                String xvj = parts[i];

                word.set(xnj + " " + classLabel + " "
+ xvj);

                if (myMap.containsKey(word.toString()))
{
                    String p =
myMap.get(word.toString());
                    double prob =
Double.parseDouble(p);

                    if (classLabel.equals("0")) {
                        zero_F *= prob;
                    } else {
                        one_F *= prob;
                    }
                } else {
                    // 朴素贝叶斯模型中，如果某个属性值在训
练集中没有出现过，那么就会导致概率为0，所以这里直接跳过
                    continue;
                }
                double sum = zero_F + one_F;
                if (zero_F / sum > one_F / sum + 0.1)
{
                    // 预测为0
                    String resultKey =
classLabel.equals("0") ? "R_0_P_0" : "R_1_P_0";
                    target.set(resultKey);
                    context.write(target, one);
                } else {
                    // 预测为1
                    String resultKey =
classLabel.equals("0") ? "R_0_P_1" : "R_1_P_1";
                    target.set(resultKey);
                    context.write(target, one);
                }
            }



            context.write(word, one);
        }
    }

    public static class LoanApplicationReducer extends
Reducer<Text, IntWritable, Text, IntWritable> {

        private IntWritable Num = new IntWritable();

        public void reduce(Text key,
Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {

            int count=0;
            for (IntWritable val : values) {
                count+=val.get();
            }
            Num.set(count);
            context.write(key, Num);
        }
    }

    public static void main(String[] args) throws
Exception {
        Configuration conf = new Configuration();
        conf.set("fs.default.name",
"hdfs://localhost:9000");
```

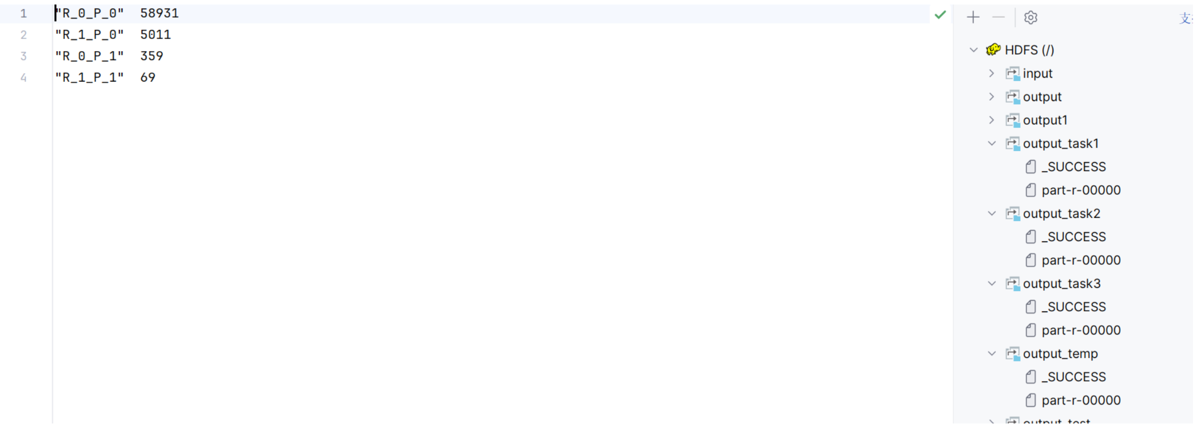
```
        Job job = Job.getInstance(conf, "Final Predict Job");
        job.setJarByClass(FinalPredict.class);


        job.setMapperClass(FinalPredict.LoanApplicationMapper.class);
        job.setCombinerClass(FinalPredict.LoanApplicationReducer.class);
        job.setReducerClass(FinalPredict.LoanApplicationReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        URI path = URI.create("hdfs://192.168.52.133:9000/output_temp/part-r-00000");
        job.addCacheFile(path);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

- 首先通过setup函数，先将预处理的数据进行缓存处理，便于后续使用。
- 然后通过MAP将测试集数据进行读取，再通过朴素贝叶斯模型原理，如果训练数据中出现了，则根据概率预测TARGET值，如果没有出现则跳过
- 最后通过统计集中预测情况进行结果分析

运行结果



```
output_task3
    _SUCCESS
    part-r-00000
```



```
"R_0_P_0"	58931
"R_1_P_0"	5011
"R_0_P_1"	359
"R_1_P_1"	69
```

朴素贝叶斯混淆矩阵：

	Predicted 0	Predicted 1
Actual 0	58931	359
Actual 1	5011	69

现在，我们可以使用这些值计算准确率（accuracy），它是正确预测的样本数占总样本数的比例：

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

根据给定的值，代入公式：

$$Accuracy = \frac{69 + 58931}{69 + 58931 + 5011 + 359} = 0.9146$$

因此我们的预测准确率达到了0.9146