

# 金融大数据实验2实验报告

211275010 谢子骐

## 一、task1

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。

1、设计思路：

在任务一中，该任务程序类似于单词计数的程序，不过不同的是统计相应列中0、1出现的个数，而不是单词登录字符串的个数

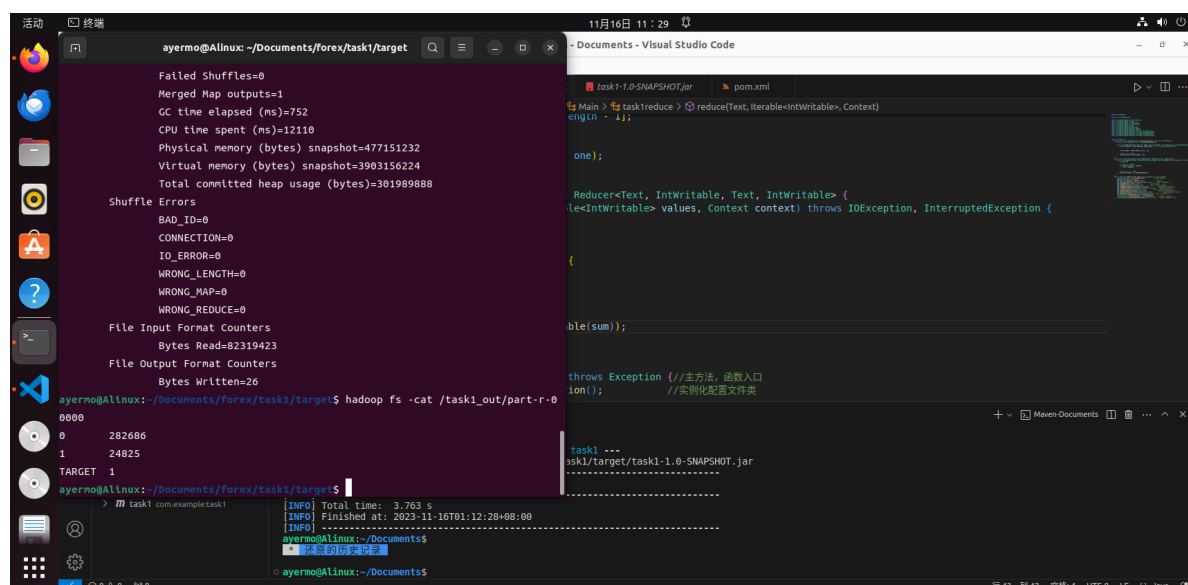
(1) map阶段：

在map阶段，程序读入一个int格式和一个string格式（key1和value1），在map函数中，将target的值作为key，将单个的数量（即1）作为value，输出给shuffle&reduce阶段。

(2) reduce阶段

在shuffle阶段，程序会对map阶段传来的键值对进行统计，输出的格式是key=target，value=<1,1,1.....>（即出现次数），因此，在reduce阶段，程序只需对相应target标签下的value数组中的1进行统计即可。

2、实验结果：



The screenshot displays a terminal window on the left and a VS Code editor on the right. The terminal shows the output of a Hadoop MapReduce job, including shuffle statistics, map output counts, and a final file listing. The VS Code editor shows the source code for task1-SNAPSHOT.jar, which implements a MapReduce program to count the number of records with a specific TARGET value (0 or 1).

```
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=752
CPU time spent (ms)=12110
Physical memory (bytes) snapshot=477151232
Virtual memory (bytes) snapshot=3903156224
Total committed heap usage (bytes)=301909888

Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0

File Input Format Counters
Bytes Read=82319423
File Output Format Counters
Bytes Written=26

ayerno@Alinux:~/Documents/forex/task1/target$ hadoop fs -cat /task1_out/part-r-0
0000
0      282686
1      24825
TARGET 1

ayerno@Alinux:~/Documents/forex/task1/target$
```

```
task1 ---
task1/target/task1-1.0-SNAPSHOT.jar
-----

Reducer<Text, IntWritable, Text, IntWritable> {
  <IntWritable> values, Context context) throws IOException, InterruptedException {

  }

  write(sum);

  throws Exception { //主方法，函数入口
    ion(); //实例化配置文件类

    task1 ---
    task1/target/task1-1.0-SNAPSHOT.jar
    -----
```

## 二、task2

## 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。

### 1、设计思路：

任务2与任务一要求相似，都是统计某一类数据的个数，唯一不同的是，任务二要求降序输出，因此在程序中需要自己设计比较函数，并在main函数运行部分，应用自己写的compare函数

```
job.setSortComparatorClass(IntWritableDecreasingComparator.class);
```

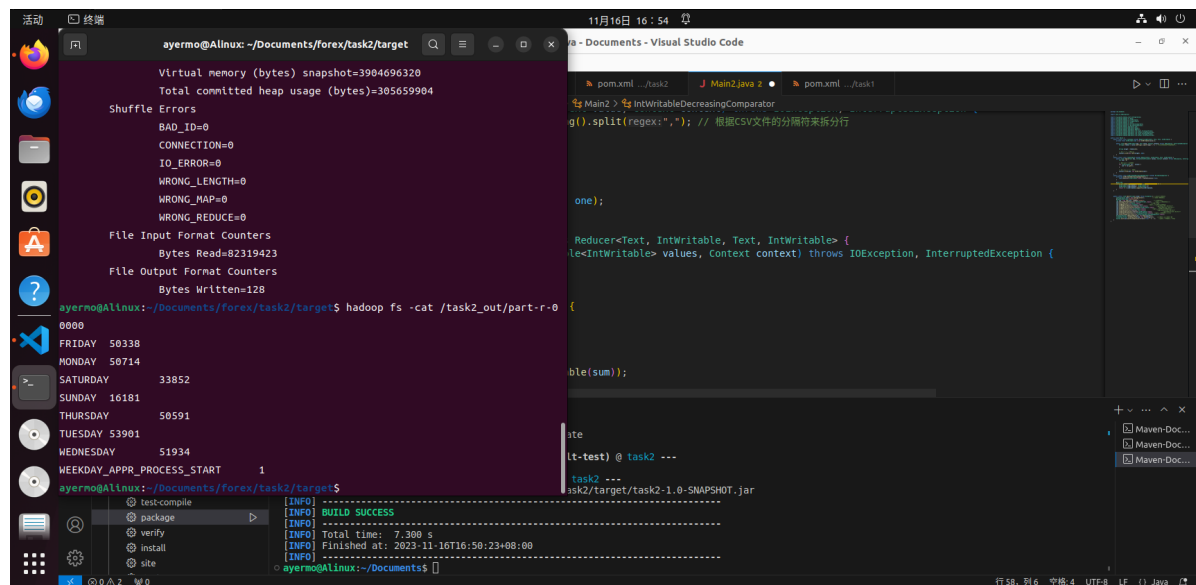
我的比较函数内容如下：

```
public static class IntWritableDecreasingComparator extends WritableComparator {  
    protected IntWritableDecreasingComparator() {  
        super(IntWritable.class, true);  
    }  
}
```

```
@Override  
public int compare(writableComparable a, writableComparable b) {  
    IntWritable intWritableA = (IntWritable) a;  
    IntWritable intWritableB = (IntWritable) b;  
    return -1 * intWritableA.compareTo(intWritableB);  
}  
}
```

即如果 $a > b$ ，则先输出a。

### 2、运行结果：



## 三、task3

在任务三中，我选择用KNN算法来进行二分类，但是KNN有一个缺点是运行过慢。

### 1、设计思路：

task3需要根据数据集选择特征来分类描述target这一列的值，于是涉及到两个方面，（1）数据集的处理，（2）特征的选择，（3）分类算法的选择，（4）分类结果的评估。

#### （1）特征的选择

根据所学习的专业知识以及日常生活的经验积累，我选择用：NAME\_CONTRACT\_TYPE  
AMT\_INCOME\_TOTAL AMT\_CREDIT NAME\_INCOME\_TYPE NAME\_EDUCATION\_TYPE  
NAME\_FAMILY\_STATUS FLAG\_CONT\_MOBILE REGION\_RATING\_CLIENT  
REG\_REGION\_NOT\_LIVE\_REGION ORGANIZATION\_TYPE OBS\_30\_CNT\_SOCIAL\_CIRCLE  
DEF\_30\_CNT\_SOCIAL\_CIRCLE OBS\_60\_CNT\_SOCIAL\_CIRCLE DEF\_60\_CNT\_SOCIAL\_CIRCLE  
FLAG\_DOCUMENT\_2 FLAG\_DOCUMENT\_3 FLAG\_DOCUMENT\_4 FLAG\_DOCUMENT\_5  
FLAG\_DOCUMENT\_6 FLAG\_DOCUMENT\_7 FLAG\_DOCUMENT\_8 FLAG\_DOCUMENT\_9  
FLAG\_DOCUMENT\_10 FLAG\_DOCUMENT\_11 FLAG\_DOCUMENT\_12 FLAG\_DOCUMENT\_13  
FLAG\_DOCUMENT\_14 FLAG\_DOCUMENT\_15 FLAG\_DOCUMENT\_16 FLAG\_DOCUMENT\_17  
FLAG\_DOCUMENT\_18 FLAG\_DOCUMENT\_19 FLAG\_DOCUMENT\_20 FLAG\_DOCUMENT\_21 作为特征数据。

其中NAME\_INCOME\_TYPE、NAME\_EDUCATION\_TYPE、NAME\_FAMILY\_STATUS、NAME\_CONTRACT\_TYPE为string类型，无法量化比较，因此对其中的值的大小进行量化，因为是分类模型，所以对值的大小没有严谨的要求，故将其中string一样的值视作相同大小的int类型进行量化，int类型的值依次从0开始叠加。最终得到处理后的新数据集application4.csv。同时，按照要求将该数据集划分为8:2的训练的测试集。分别为train.csv and test.csv因为数据集过大有30w条数据，在程序运行了1h之后，依然是进度map: 1%，reduce: 0%。故选择将测试集合进行一定程度的删除。我在进行了各种尝试之后，为了保证实验结果能够在有限时间内顺利得到，最终将train和test两个文件确定为：46006条和11501条。

## (2) 数据处理

在选择相应的特征之后，需要对相应的特征数据进行处理。其中NAME\_INCOME\_TYPE、NAME\_EDUCATION\_TYPE、NAME\_FAMILY\_STATUS、NAME\_CONTRACT\_TYPE为string类型，无法量化比较，因此对其中的值的大小进行量化，因为是分类模型，所以对值的大小没有严谨的要求，故将其中string一样的值视作相同大小的int类型进行量化，int类型的值依次从0开始叠加。  
FLAG\_DOCUMENT\_2 FLAG\_DOCUMENT\_3 FLAG\_DOCUMENT\_4 FLAG\_DOCUMENT\_5  
FLAG\_DOCUMENT\_6 FLAG\_DOCUMENT\_7 FLAG\_DOCUMENT\_8 FLAG\_DOCUMENT\_9  
FLAG\_DOCUMENT\_10 FLAG\_DOCUMENT\_11 FLAG\_DOCUMENT\_12 FLAG\_DOCUMENT\_13  
FLAG\_DOCUMENT\_14 FLAG\_DOCUMENT\_15 FLAG\_DOCUMENT\_16 FLAG\_DOCUMENT\_17  
FLAG\_DOCUMENT\_18 FLAG\_DOCUMENT\_19 FLAG\_DOCUMENT\_20 FLAG\_DOCUMENT\_21有一定数量的缺失值，因此用0补填空缺项。之后我的mapreduce程序在读取csv文件时，没有设置第一行列名要单独读，因此在处理数据时，要将列名这一行去除。

## (3) 分类算法的选择

在诸多分类算法中，因为我对KNN算法比较了解，故选择KNN算法。我的KNN编程思路如下：

因为之前有所了解，知道KNN算法在大数据环境下运行起来较为缓慢，因此在进行编程前我便在网上查提高效率的方法。在一篇文章中提到：在map后、reduce前，对程序进行一个combiner过程，即对map完成的数据进行K临近选择，将选择后的数据再传给reduce阶段进行reduce。

预处理阶段：将test文件上传至缓存区，以备在计算距离时用到。

map阶段：读入一个训练样本，将标签（label）提取出来，将属性变成一个Double类型的数组（train\_point）；将测试样本逐个转变成Double类型的数组（test\_point），使用定义好的计算距离方法计算一下距离distance。将测试样本在测试集中的行偏移量作为键key（此处就是数组中的索引），距离加分隔符再加上标签作为value。也就是（股票ID， distance@label（value））。

combiner阶段：这里采用treemap形式存储同一测试样本下，与训练样本的距离和标签。其中距离作为map的键，标签作为值。Treemap会在形成的过程中自动对键key排序，默认是升序，正好符合我们对距离小的排在前面的要求。将前K项按照刚才的格式输出。

reduce阶段：首先和Combiner的阶段一样，先排序，再找到前K项最近的，但是这里多了一步对前K项标签进行计数统计，找出数量最多的标签为测试样本的标签。

(4) 预测结果的测评，我选择用准确度accuracy指标来衡量预测结果的准确度，由于不太习惯java的环境，于是我选择在拿到预测结果的数据后，在python下进行预测指标的计算。

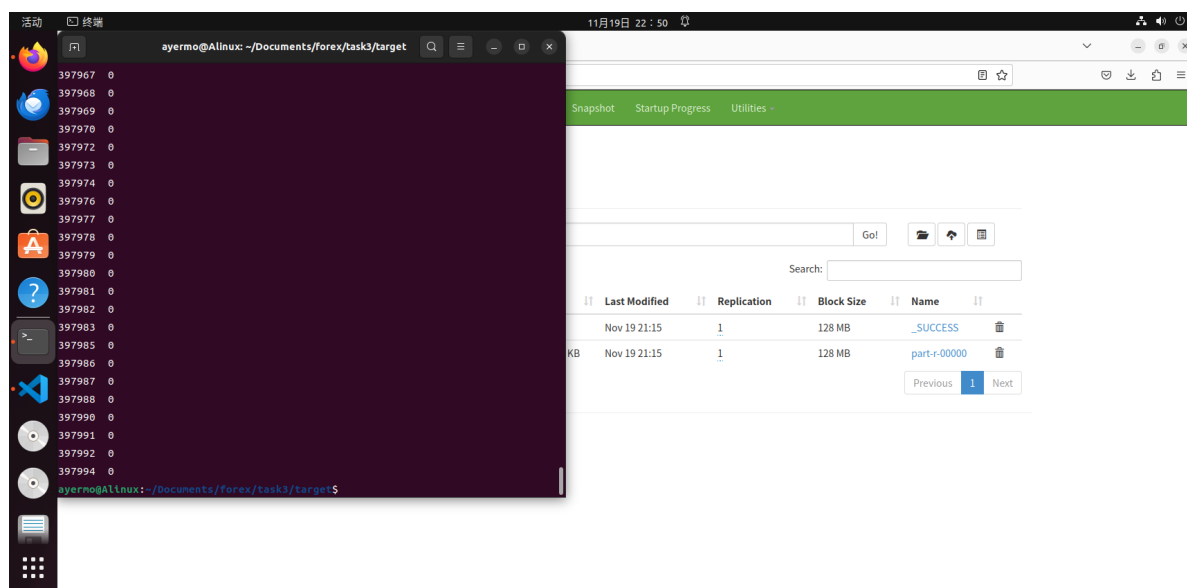
## 2、实验结果：

### (1) 处理数据：

```
PS D:\Forexp\forP> & D:/python/python.exe d:/Forexp/forP/shockcode.py
The CSV file has 246008 columns.
PS D:\Forexp\forP> & D:/python/python.exe d:/Forexp/forP/shockcode.py
The CSV file has 246007 columns.
PS D:\Forexp\forP> & D:/python/python.exe d:/Forexp/forP/shockcode.py
The CSV file has 61502 columns.
PS D:\Forexp\forP> & D:/python/python.exe d:/Forexp/forP/shockcode.py
The CSV file has 246007 columns.
The CSV file has 46007 columns.
```

### (2) 预测阶段：

我的预测结果如下（局部）：



### (3) 对结果进行准确性分析：

将hadoop的运行结果文件part-r-0000文件和test文件一并导入python中存为dataframe格式，之后先用merge函数对股票代码这一列进行合并，统计每一行中股票代码对应的两项的数据是否相同，统计相同的个数。

统计分析的python代码如下：

```
# 导入tushare
import tushare as ts
import datetime
import time
from datetime import timedelta
from dateutil.relativedelta import *
import pandas as pd
import numpy as np
from pandas import Series
from sqlalchemy import create_engine
import mysql.connector
# 连接MySQL数据库
file_path="D:\其他\part-r-00000(1)"
df1 = pd.read_csv(file_path, delimiter='\t', header=None)
df2 = pd.read_csv('D:/DesktopD/金融大数据/test_data6.csv', header=None)
```

```
#print(df1)
print("_____")
#print(df2)
merged = pd.merge(df1, df2, on=0)

# 统计相同数据的个数
same_count = (merged['1_x'] == merged['1_y']).sum()

print(f"相同的数据个数为: {same_count}")
```

最后得到数据

```
PS D:\Forexp\forP> & D:/python/python.exe d:/Forexp/forP/paper.py
相同的数据个数为: 10580
PS D:\Forexp\forP>
```

即在11501条数据中，有10580个数据是相同的，因此准确率 $accuracy=10580/11501=91.1\%$

结果较为准确。

3、总结：

选择NAME\_CONTRACT\_TYPE AMT\_INCOME\_TOTAL AMT\_CREDIT NAME\_INCOME\_TYPE  
NAME\_EDUCATION\_TYPE NAME\_FAMILY\_STATUS FLAG\_CONT\_MOBILE  
REGION\_RATING\_CLIENT REG\_REGION\_NOT\_LIVE\_REGION ORGANIZATION\_TYPE  
OBS\_30\_CNT\_SOCIAL\_CIRCLE DEF\_30\_CNT\_SOCIAL\_CIRCLE OBS\_60\_CNT\_SOCIAL\_CIRCLE  
DEF\_60\_CNT\_SOCIAL\_CIRCLE FLAG\_DOCUMENT\_2 FLAG\_DOCUMENT\_3 FLAG\_DOCUMENT\_4  
FLAG\_DOCUMENT\_5 FLAG\_DOCUMENT\_6 FLAG\_DOCUMENT\_7 FLAG\_DOCUMENT\_8  
FLAG\_DOCUMENT\_9 FLAG\_DOCUMENT\_10 FLAG\_DOCUMENT\_11 FLAG\_DOCUMENT\_12  
FLAG\_DOCUMENT\_13 FLAG\_DOCUMENT\_14 FLAG\_DOCUMENT\_15 FLAG\_DOCUMENT\_16  
FLAG\_DOCUMENT\_17 FLAG\_DOCUMENT\_18 FLAG\_DOCUMENT\_19 FLAG\_DOCUMENT\_20  
FLAG\_DOCUMENT\_21作为特征解释变量对target项进行预测的准确率较高，为91.1%。