

# 实验二 贷款违约预测

211275007 康钊源

## 一、设计思路

- 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。

只需要统计数据中违约与未违约的数量即可，即对target的两种不同的键计数即可

- 任务二：

编写MapReduce程序，统计一周当中每天申请贷款的交易数

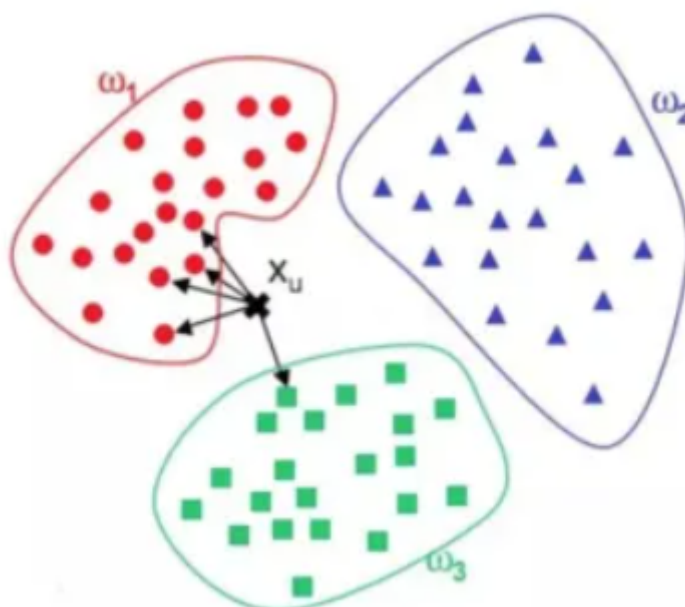
WEEKDAY\_APPR\_PROCESS\_START，并按照交易数从大到小进行排序。

与任务一类似，只是键的类型由0.1两种变为了七种不同的星期。但需注意输出要按交易数从大到小进行排序，我使用TreeMap数据结构实现了这一功能。

- 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。

最为复杂的一问。选择使用课上所讲的KNN分类方法，这需要我们首先选取一些属性，并将其化为特征向量。在对预测样本进行预测时，只需要找到训练集中特征向量与待预测样本的特征向量欧氏距离最近的K个点，并选择其中权重最大的那种目标属性作为该样本的预测值。原理如下图：



而评估指标则选取最为常见的准确率（Accuracy），精确率（Precision），召回率（Recall），F1-score。他们的计算如下：

		实际结果	
		1	0
预测结果	1	TP	FP
	0	FN	TN

- TP: 预测为1, 实际为1, 预测正确。
- FP: 预测为1, 实际为0, 预测错误。
- FN: 预测为0, 实际为1, 预测错误。
- TN: 预测为0, 实际为0, 预测正确。

准确率 = (TP+TN)/total

精确率 = TP/(TP+FP)

召回率 = TP/(TP+FN)

F1-score = 2 \* 精确率 \* 召回率 / (精确率 + 召回率)

#### • 文件结构

用maven组织项目，程序文档有：

主类Driver.java

任务一DefaultCountMapper.java、DefaultCountReducer.java

任务二DailyLoanCountMapper.java、DailyLoanCountReducer.java

任务三KNN\_Mapper.java、KNN\_Reducer.java、MyData.java

## 二、数据预处理

为了方便mapreduce的读取及处理，对数据进行如下预处理：

- 1.删除表格的第一行，因为在map阶段不方便区分是数据还是列名，不如直接把列名行删去
- 2.手动将原表格按照时间先后8:2拆分成训练集与测试集，并直接在训练集与测试集中保留需要用到的属性

## 三、程序实现

### (1) 主类

完成输入的接收及输出的设置，分别配置任务一到任务三的作业类。

```

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver {
    Run | Debug
    public static void main(String[] args) throws Exception {
        if (args.length != 6) {
            System.err.println(x:"Usage: Driver <input path><training data><test data> <output path1> <output path2> <output path3>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();

        // 任务一 - 统计违约和非违约的数量
        Job job1 = Job.getInstance(conf, jobName:"Default Count");
        job1.setJarByClass(cls:Driver.class);
        job1.setMapperClass(cls:DefaultCountMapper.class);
        job1.setReducerClass(cls:DefaultCountReducer.class);
        job1.setOutputKeyClass(theClass:Text.class);
        job1.setOutputValueClass(theClass:IntWritable.class);
        FileInputFormat.addInputPath(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[3]));

        // 任务二 - 统计每天申请贷款的交易数
        Job job2 = Job.getInstance(conf, jobName:"Daily Loan Count");
        job2.setJarByClass(cls:Driver.class);
        job2.setMapperClass(cls:DailyLoanCountMapper.class);
        job2.setReducerClass(cls:DailyLoanCountReducer.class);

```

值得注意的是在job3的配置中还配置了KNN算法的一些参数（如K值，向量维度等）以及计算评估指标将用到的值。

```

// 任务三 - 基于贷款数据建立违约检测模型

conf.setInt(name:"K",value:5);//设置KNN算法的K值
conf.setInt(name:"testDataNum",value:0);//设置全局计数器，记录测试数据数目
conf.setInt(name:"dimension",value:8);//设置向量维度
conf.setInt(name:"TP",value:0);
conf.setInt(name:"TN",value:0);
conf.setInt(name:"FP",value:0);
conf.setInt(name:"FN",value:0);
Job job3 = Job.getInstance(conf, jobName:"Default Detection");
job3.setJarByClass(cls:Driver.class);
job3.setMapperClass(cls:KNN_Mapper.class);
job3.setReducerClass(cls:KNN_Reducer.class);
job3.setMapOutputKeyClass(theClass:LongWritable.class);
job3.setMapOutputValueClass(theClass:Text.class);
job3.setOutputKeyClass(theClass:LongWritable.class);
job3.setOutputValueClass(theClass:Text.class);
job3.addCacheFile(new Path(args[1]).toUri());

FileInputFormat.addInputPath(job3, new Path(args[2]));
FileOutputFormat.setOutputPath(job3, new Path(args[5]));
// 提交作业并等待完成
boolean job1Completed = job1.waitForCompletion(verbose:true);
boolean job2Completed = job2.waitForCompletion(verbose:true);
boolean job3Completed = job3.waitForCompletion(verbose:true);

```

## (2) 任务一

mapper:

如下图，只需将target作为键，1作为值输出即可

```

package com.example;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class DefaultCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(value:1);
    private Text label = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] tokens = value.toString().split(regex:"");
        // 获取TARGET标签
        String target = tokens[59];
        label.set(target);
        context.write(label, one);
    }
}

```

reducer:

将两种键的值分别求和，即能得到分别的统计数。

```

package com.example;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class DefaultCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private IntWritable result = new IntWritable();

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

```

### (3) 任务二

mapper:

类似任务一

```

package com.example;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.io.LongWritable;

public class DailyLoanCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(value:1);
    private Text weekday = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] tokens = value.toString().split(regex:"");
        // 获取WEEKDAY_APPR_PROCESS_START字段
        String startDay = tokens[25];
        weekday.set(startDay);
        context.write(weekday, one);
    }
}

```

reducer:

与任务一最大的不同在于多了对值进行排序，故将统计求和作为值，星期作为键放入TreeMap数据结构，实现了按键的大小自动排序，最后再键值对反过来输出即可。

```

import java.util.TreeMap;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.util.Collections;

public class DailyLoanCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    private TreeMap<Integer, String> resultMap;

    public void setup(Context context) {
        resultMap = new TreeMap<>(Collections.reverseOrder());
    }

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
        resultMap.put(sum, key.toString());
    }

    public void cleanup(Context context) throws IOException, InterruptedException {
        for (Map.Entry<Integer, String> entry : resultMap.entrySet()) {
            context.write(new Text(entry.getValue()), new IntWritable(entry.getKey()));
        }
    }
}

```

#### (4) 任务三

Mydata.java:

为了方便进行KNN方法中的数据处理，设计了这个类

主要实现了

- 直接从数据中提取目标属性，向量维度

- 计算两个特征向量数据间的欧式距离

```
import java.util.Vector;

public class MyData {

    //向量维度
    private Integer dimension;
    //向量坐标
    private Vector<Double>vec = new Vector<Double>();
    //target属性
    private String attr;

    public void setAttr(String attr)
    {
        this.attr = attr;
    }

    public void setVec(Vector<Double> vec) {
        this.dimension = vec.size();
        for(Double d : vec)
        {
            this.vec.add(d);
        }
    }

    public double calDist(MyData data1)//计算两条数据之间的欧式距离
    {
        try{
            if(this.dimension != data1.dimension)
                throw new Exception(message:"These two vectors have different dimensions.");
        }
        catch (Exception e)
        {
            System.out.println(e.getMessage());
            System.exit(-1);
        }
        double dist = 0;
        for(int i = 0;i<dimension;i++)
        {
            dist += Math.pow(this.vec.get(i)-data1.vec.get(i),2);
        }
        dist = Math.sqrt(dist);
        return dist;
    }

    public String getAttr() {
        return attr;
    }
}
```

mapper:

任务三主要就是靠mapper实现了KNN算法，首先在类中进行一些变量设置，读取训练集数据并将value分成多个维度的特征，同时设置每条数据对应的目标属性。

```
import java.io.*;

public class KNN_Mapper extends Mapper<LongWritable, Text, LongWritable, Text> {
    private Text text = new Text(); // 输出Val值

    private LongWritable longWritable = new LongWritable();
    private LongWritable TP = new LongWritable();
    private LongWritable FP = new LongWritable();
    private LongWritable TN = new LongWritable();
    private LongWritable FN = new LongWritable();
    private Integer K; // K值

    private Configuration conf; // 全局配置
    private Integer dimension; // 维度
    private List<MyData> training_data = new ArrayList<>();

    private void readTrainingData(URL uri) // 读取训练数据到training_data中
    {
        System.err.println(x: "Read Training Data");
        try {
            Path patternsPath = new Path(uri.getPath());
            String patternsFileName = patternsPath.getName().toString();
            BufferedReader reader = new BufferedReader(new FileReader(
                patternsFileName));
            String line;
            Vector<Double> vec = new Vector<>();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

        String line;
        Vector<Double>vec = new Vector<>();
        while ((line = reader.readLine()) != null) {

            String[] strings = line.split(regex:",");

            for(int i=0;i<dimension;i++)
            {
                vec.add(Double.valueOf(strings[i]));
            }
            MyData myData = new MyData();
            myData.setVec(vec);
            myData.setAttr(strings[dimension]);
            training_data.add(myData);
            vec.clear();
        }
        reader.close();
    }
    catch (FileNotFoundException e)
    {
        e.printStackTrace();
    }
    catch (IOException e)
    {
        e.printStackTrace();
    }
    System.err.println(x:"Read End");
}

```

下面这部分代码实现了计算欧氏距离权重，以及读取一条测试集数据。



```

private double Gaussian(double dist)
{
    //a = 1,b=0,c = 0.9,2*c^2 = 1.62
    double weight = Math.exp(-Math.pow(dist,b:2)/(1.62));
    return weight;
}

@Override
public void setup(Context context) throws IOException,
    InterruptedException {

    conf = context.getConfiguration();
    this.K = conf.getInt(name:"K",defaultValue:1);
    this.dimension = conf.getInt(name:"dimension",defaultValue:1);
    URI[] uri = context.getCacheFiles();
    readTrainingData(uri[0]);
}

@Override
public void map(LongWritable key, Text value, Context context
) throws IOException, InterruptedException {
    String line = value.toString();

    try {
        String[] strings = line.split(regex:",");

        if (strings .length!=dimension+1) {

```

将这条测试集数据的特征向量与所有训练集数据进行比较，求出其中距离最近的K条数据。

```

    }
    vec.remove(vec.size() - 1);
    MyData testData = new MyData();
    testData.setVec(vec);

    //计算与样本的K近邻

    //存放K近邻的优先级队列，元素类型为<距离，属性>
    PriorityQueue<Pair<Double,String>>K_nearest = new PriorityQueue<>((a,b)->(a.getKey()-b.getKey())?-1:1);
    double dist;
    for(MyData data : this.training_data)
    {
        dist = testData.calDist(data);
        if(K_nearest.size()<this.K)
        {
            K_nearest.add(new Pair<>(dist,data.getAttr()));
        }
        else{
            if(dist < K_nearest.peek().getKey())
            {
                K_nearest.poll();
                K_nearest.add(new Pair<>(dist,data.getAttr()));
            }
        }
    }
}

```

之后计算这K个点中权重最大的目标属性，并将其作为该条测试数据的预测值。

```
//获取到K近邻后，通过高斯函数处理每条数据，并累加相同属性的权值，通过Hash_table实现
Hashtable<String,Double>weightTable = new Hashtable<>();
while(!K_nearst.isEmpty())
{
    double d = K_nearst.peek().getKey();
    String attr = K_nearst.peek().getValue();
    double w = this.Gaussian(d);
    if(!weightTable.containsKey(attr))
    {
        weightTable.put(attr,w);
    }
    else{
        weightTable.put(attr,weightTable.get(attr)+w);
    }
    K_nearst.poll();
}

//选取权重最大的标签作为输出
Double max_weight = Double.MIN_VALUE;
String predict_attr = "";

for(Iterator<String> itr = weightTable.keySet().iterator();itr.hasNext();){
    String hash_key = (String)itr.next();
    Double hash_val = weightTable.get(hash_key);
    if(hash_val > max_weight)
    {
```

之后将预测值与真实值比较，得出该条预测数据真/假阳/阴性，并计入相对应的统计指标。

同时将样本序号作为键，预测结果作为值写入context。

```
context.put(predict_attr);
//
TP.set(conf.getLong(name:"TP",defaultValue:0));
FP.set(conf.getLong(name:"FP",defaultValue:0));
TN.set(conf.getLong(name:"TN",defaultValue:0));
FN.set(conf.getLong(name:"FN",defaultValue:0));
if (predict_attr.equals(anObject:"1") && true_attr.equals(anObject:"1"))
    conf.setLong(name:"TP", TP.get() + 1);
if (predict_attr.equals(anObject:"1") && true_attr.equals(anObject:"0"))
    conf.setLong(name:"FP", FP.get() + 1);
if (predict_attr.equals(anObject:"0") && true_attr.equals(anObject:"0"))
    conf.setLong(name:"TN", TN.get() + 1);
if (predict_attr.equals(anObject:"0") && true_attr.equals(anObject:"1"))
    conf.setLong(name:"FN", FN.get() + 1);
//获取测试数据条数，用作下标计数
longWritable.set(conf.getLong(name:"testDataNum",defaultValue:0));
conf.setLong(name:"testDataNum",longWritable.get()+1);//计数加一
context.write(longWritable,text);
float total = (float) longWritable.get();
```

当统计完最后一条数据时，根据之前所得到的真/假阳/阴性统计指标计算准确率（Accuracy），精确率（Precision），召回率（Recall），F1-score这四个评估指标。

```

float total = (float) longWritable.get();
if(total==61245)
{
    float tp = (float) TP.get();
    float tn = (float) TN.get();
    float fp = (float) FP.get();
    float fn = (float) FN.get();
    // 输出评估结果
    float accuracy = (tp+tn)/total;
    float precision = tp/(tp+fp);
    float recall = tp/(tp+fn);
    float f1 = 2*precision*recall/(precision+recall);
    System.out.println("Total: " + total);
    System.out.println("Accuracy: " + accuracy);
    System.out.println("Precision: " + precision);
    System.out.println("Recall: " + recall);
    System.out.println("F1-score: " + f1);
}
}
catch (Exception e) {
    System.err.println(e.toString());
    System.exit(-1);
}
}

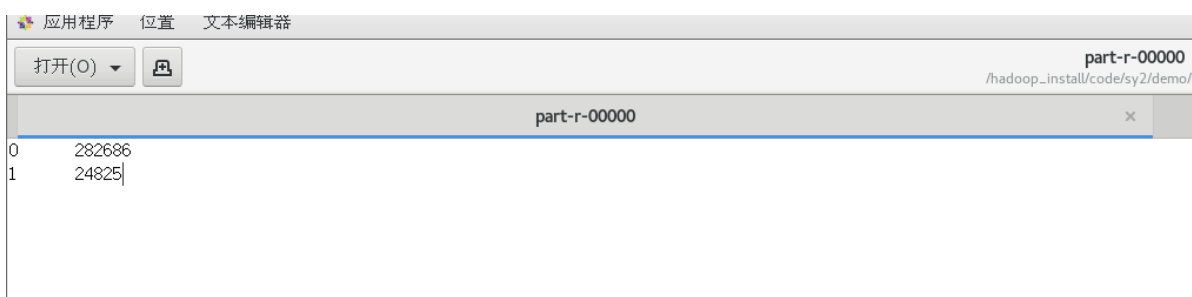
```

reducer:

本来预备在reducer中计算评估指标，但发现不太可行，故只在reducer中传递mapper传来的键值对。

## 四、运行结果

### • 任务一

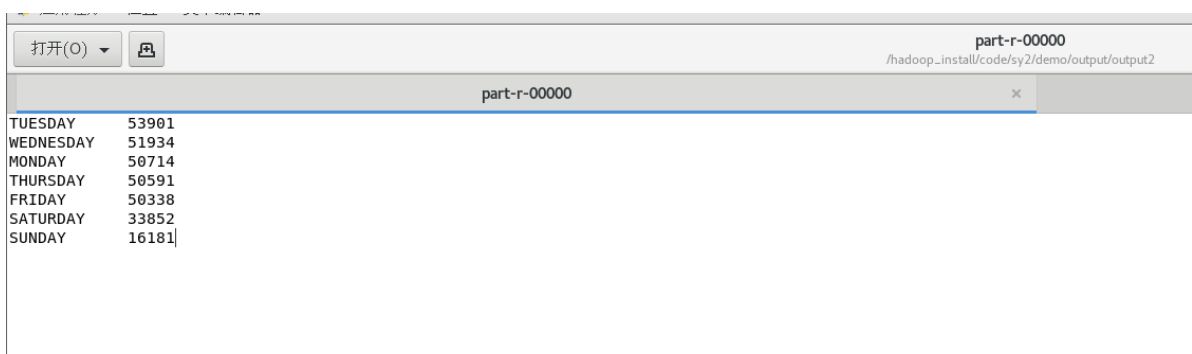


key	value
0	282686
1	24825

输出结果如上图，经检验为正确的

可以看到未违约和违约的比例大致是92:8，这个比例使得在任务三中accuracy的大小不是那么重要

### • 任务二

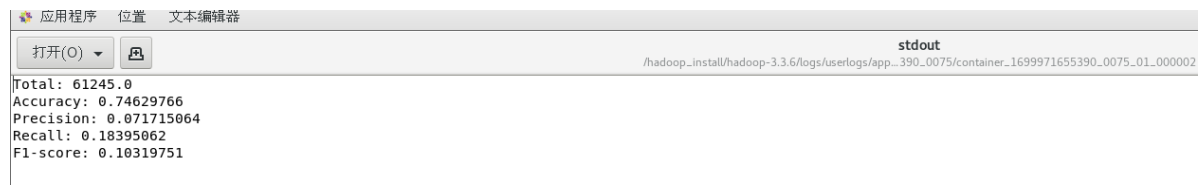


key	value
TUESDAY	53901
WEDNESDAY	51934
MONDAY	50714
THURSDAY	50591
FRIDAY	50338
SATURDAY	33852
SUNDAY	16181

输出结果如预期一样按交易数量从大到小进行了排序

可以发现一个有趣的现象——贷款行为多发生于周中，而周末发生较少

- 任务三



```
应用程序 位置 文本编辑器
打开(O) [icon] stdout
/hadoop_install/hadoop-3.3.6/logs/userlogs/app_390_0075/container_1699971655390_0075_01_0000002

Total: 61245.0
Accuracy: 0.74629766
Precision: 0.071715064
Recall: 0.18395062
F1-score: 0.10319751
```

上图为任务三KNN模型的评估指标，准确率为74.6%但因为样本偏向性太大，所以这个指标意义不大

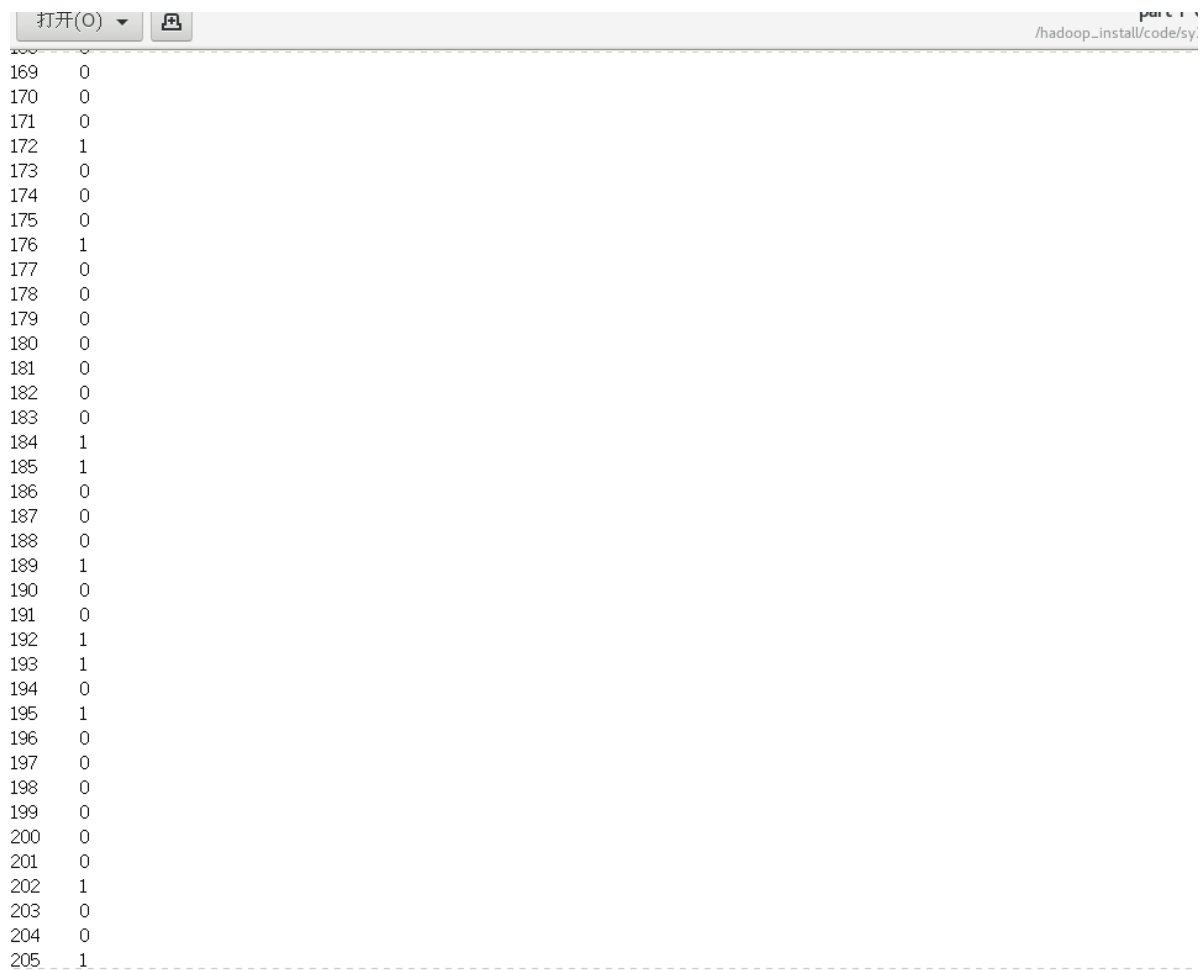
而精确率为7.17%，召回率为18.4%，F1-score为10.3%，这些指标并不是很理想，这可能是由于

1.选择的特征维度较少（8），且可能不是最合适的用来预测的特征

2.KNN模型的K值设置得有些小

3.KNN模型本身较为简单，不适用较为复杂的贷款违约预测问题

除了评估指标外，程序也输出了所有预测样本的预测值

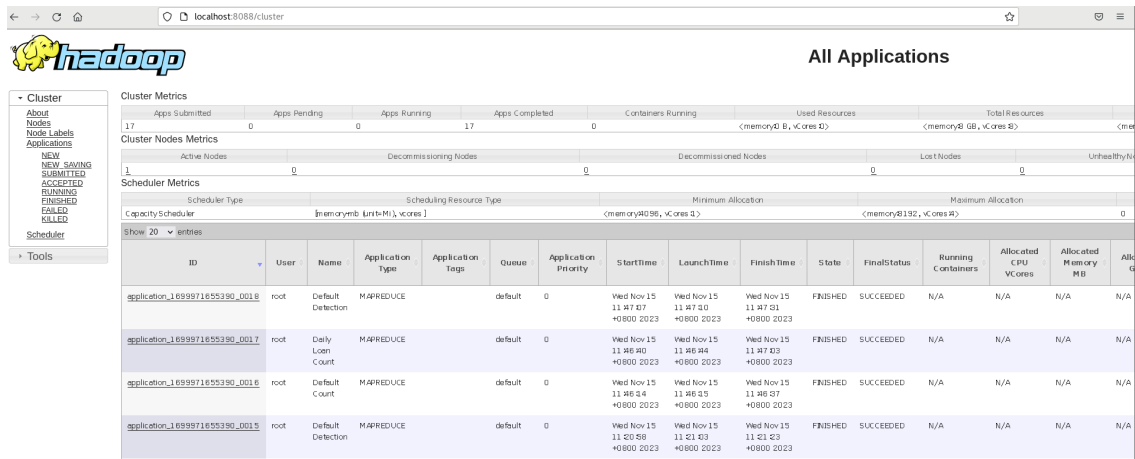


```
打开(O) [icon] stdout
/hadoop_install/code/sy

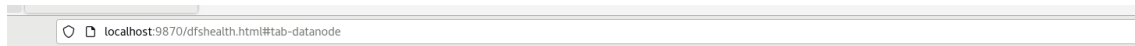
169 0
170 0
171 0
172 1
173 0
174 0
175 0
176 1
177 0
178 0
179 0
180 0
181 0
182 0
183 0
184 1
185 1
186 0
187 0
188 0
189 1
190 0
191 0
192 1
193 1
194 0
195 1
196 0
197 0
198 0
199 0
200 0
201 0
202 1
203 0
204 0
205 1
```

- Web UI

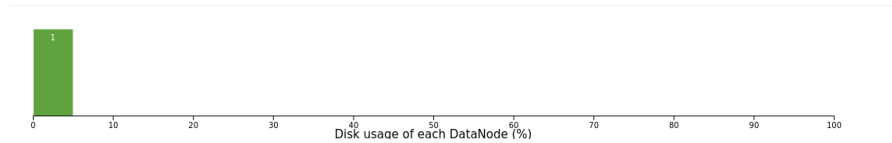
8088:



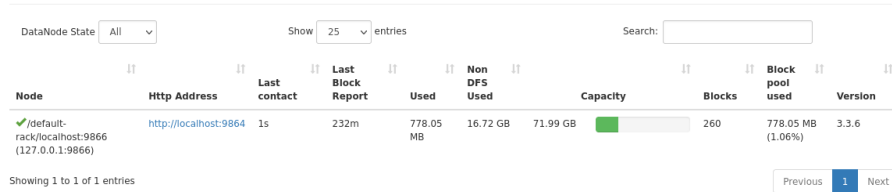
9870:



Datanode usage histogram



In operation



## 五、可改进之处

- 1. 评估指标是在map阶段输出的，不是非常符合一般常规做法，可以尝试在reducer中写一个cleanup输出评估指标。
- 2. 根据任务三的评估指标，评估效果不是很理想，根据上一部分提出的不足，可以有如下方向的修改：
  - (1) 尝试改变特征向量，可以增加维度，或选择一些其他的变量。
  - (2) 增大K。
  - (3) 选用其他分类模型，如决策树模型、朴素贝叶斯模型。
- 3. 代码运行非常耗时（近一个小时，如下图），这是容易理解的——因为每个测试集数据都要与所有训练集数据计算欧氏距离，也许可以采取增加节点，运用服务器计算资源或者优化冗余步骤来减少运行时间

---

2023-11-16	23:32:24,981	INFO	mapreduce.Job:	map	42%	reduce	0%
2023-11-16	23:33:13,584	INFO	mapreduce.Job:	map	43%	reduce	0%
2023-11-16	23:34:00,981	INFO	mapreduce.Job:	map	44%	reduce	0%
2023-11-16	23:34:43,458	INFO	mapreduce.Job:	map	45%	reduce	0%
2023-11-16	23:35:32,010	INFO	mapreduce.Job:	map	46%	reduce	0%
2023-11-16	23:36:38,550	INFO	mapreduce.Job:	map	47%	reduce	0%
2023-11-16	23:37:20,891	INFO	mapreduce.Job:	map	48%	reduce	0%
2023-11-16	23:38:03,235	INFO	mapreduce.Job:	map	49%	reduce	0%
2023-11-16	23:38:45,592	INFO	mapreduce.Job:	map	50%	reduce	0%
2023-11-16	23:39:27,992	INFO	mapreduce.Job:	map	51%	reduce	0%
2023-11-16	23:40:15,442	INFO	mapreduce.Job:	map	52%	reduce	0%
2023-11-16	23:40:57,882	INFO	mapreduce.Job:	map	53%	reduce	0%
2023-11-16	23:41:46,299	INFO	mapreduce.Job:	map	54%	reduce	0%
2023-11-16	23:42:28,672	INFO	mapreduce.Job:	map	55%	reduce	0%
2023-11-16	23:43:22,281	INFO	mapreduce.Job:	map	56%	reduce	0%
2023-11-16	23:44:16,903	INFO	mapreduce.Job:	map	57%	reduce	0%
2023-11-16	23:45:05,327	INFO	mapreduce.Job:	map	58%	reduce	0%
2023-11-16	23:45:52,825	INFO	mapreduce.Job:	map	59%	reduce	0%
2023-11-16	23:46:47,415	INFO	mapreduce.Job:	map	60%	reduce	0%
2023-11-16	23:47:35,992	INFO	mapreduce.Job:	map	61%	reduce	0%
2023-11-16	23:48:23,584	INFO	mapreduce.Job:	map	62%	reduce	0%
2023-11-16	23:49:06,159	INFO	mapreduce.Job:	map	63%	reduce	0%
2023-11-16	23:49:47,581	INFO	mapreduce.Job:	map	64%	reduce	0%
2023-11-16	23:50:36,205	INFO	mapreduce.Job:	map	65%	reduce	0%

---