1. **实验内容：**
2. **实验介绍：**

由于我们小组都没选上人工智能课程，但是我们都对人工智能很感兴趣，所以此次我们选择了人工智能领域中著名的**朴素贝叶斯算法**，将传统的朴素贝叶斯算法移植到hadoop集群环境中，实现了MapReduce 框架下的朴素贝叶斯算法，提高了算法的并行度，从而加快了算法的运行速度。对于海量数据的处理，MapReduce框架下的朴素贝叶斯算法将比传统的单机算法运行速度更快。

1. **算法介绍：**

朴素贝叶斯算法的思想基础是：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。朴素贝叶斯分类的正式定义如下：

1. 设x = { a1 , a2, …, am}为一个待分类项，而每个a为x的一个特征属性。
2. 有类别集合 C = {y1, y2, …, yn }
3. 计算 P(y1 | x), P(y2 | x), …, P(yn | x)
4. 如果 P(yk | x) = max { P(y1 | x), P(y2 | x), …, P(yn | x) }, 则 x∈yk。

为了计算第(3)步中的各个条件概率,我们应该：

1. 找到一个已知分类的待分类项集合，这个集合叫做训练样本集。
2. 统计得到在各类别下各个特征属性的条件概率估计。即

http://latex.codecogs.com/gif.latex?P(a_1|y_1),P(a_2|y_1),...,P(a_m|y_1);P(a_1|y_2),P(a_2|y_2),...,P(a_m|y_2);...;P(a_1|y_n),P(a_2|y_n),...,P(a_m|y_n)。

1. 如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：



因为分母对于所有类别为常数，所以我们只要将分子最大化皆可。又因为各特征属性是条件独立的，所以有：



**二、 实验过程：**

1. **实验环境：**

本次实验基于hadoop 2.6.0 和 jdk 1.8.0，使用由三台虚拟机组建而成的集群；

1. **测试数据：**

本次实验数据来源于美国加州大学欧文分校(UCI)的机器学习仓库，可从网址：<http://archive.ics.uci.edu/ml/datasets/Adult> 下载得到。该数据为1994年人口普查数据的一部分，预测任务是确定一个人的年收入是否超过50K。实验数据中，每一行为一条记录，每一条记录的最后一个单词为该记录所属分类，其余为该记录的属性值。每一条记录具有14个属性，具体属性含义如下：



我们对原始数据进行了一定处理，将原来的逗号分隔改为制表符分隔，并将每一行最后的’.’去掉，同时分割出测试集中的一部分放入数据集中，以解决测试集过大的问题。最后使用的数据如下：

训练集：adult\_data.txt，大小4.79M， 共42561行；

测试集：adult\_test.txt，大小723K，共6281行；

1. **代码文件功能：**

|  |  |
| --- | --- |
| 代码文件 | 作用 |
| NaiveBayesClass.java | 对测试集中的分类进行统计，得出分类的类别和各个分类的个数 |
| NaiveBayesAttribute.java | 对测试集中的各属性进行统计，得出各个属性在不同分类下的个数 |
| NaiveBayesPro.java | 结合之前的输出，统计得出各类别下各个特征属性的条件概率，输出概率表 |
| NaiveBayesTest.java | 结合概率表，通过计算预测出测试集各条记录所属的分类，并与真实答案相比较，确定预测结果正确与否 |
| NaiveBayesCount.java | 统计预测正确和预测错误的个数 |

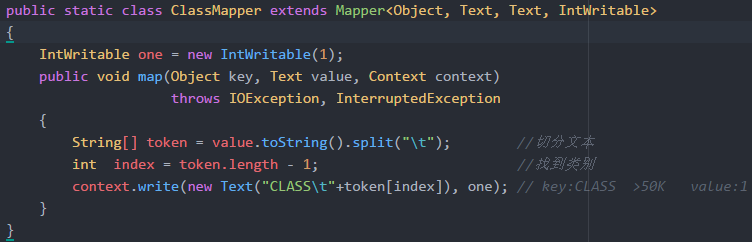
1. **代码详解：**

* Main：主要是设置各个参数，由于各个java文件的main函数类似，所以之后如无特别之处，不再展示各java文件的main函数

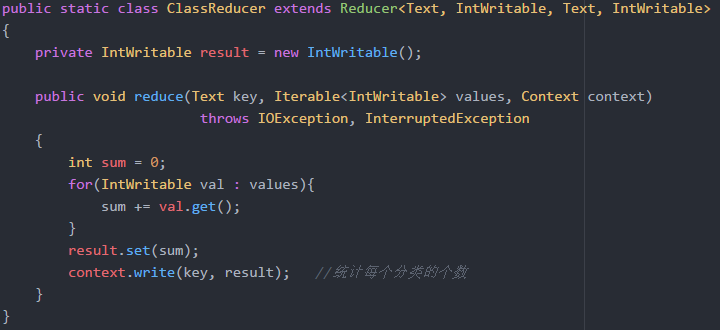


* NaiveBayesClass.java：

Mapper：把每一行记录进行切分，将分类作为key，value设为1，

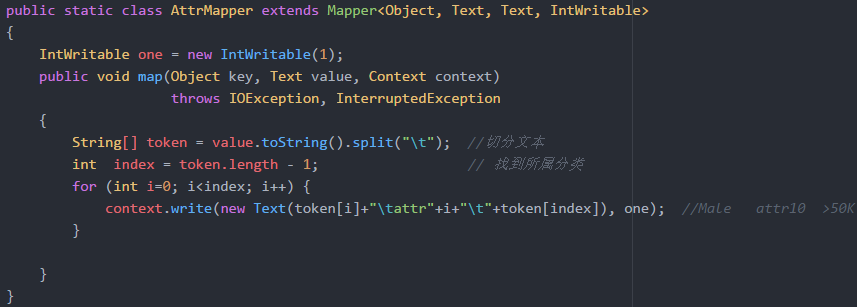


Reducer：统计各个分类个数

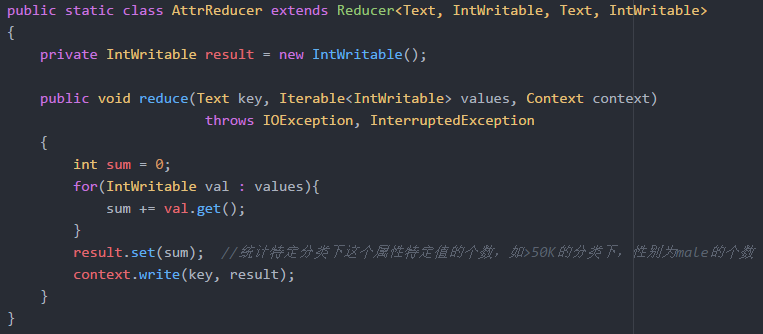


* NaiveBayesAttribute.java：

Mapper：切分出各个属性，由于属性中有多个为数值类型，所以为避免混淆，在属性值之后加上该属性为第几个属性，如”attr10”，然后加上此时该属性的所属分类作为key，value为1

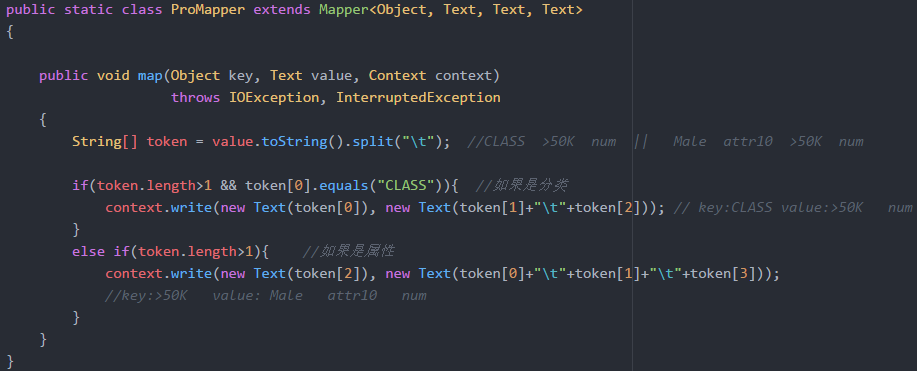


Reducer：统计不同分类下不同属性的个数

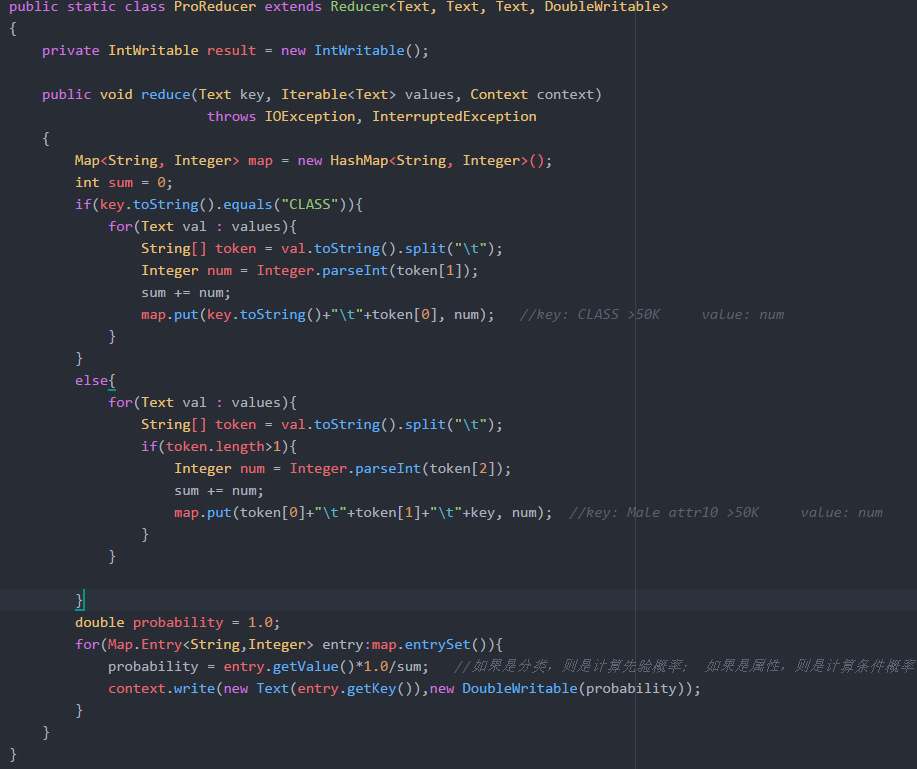


* NaiveBayesPro.java

Mapper：对文本进行切分，不同类型的数据进行不同操作，如果是分类，则key为CLASS，value为该分类和个数；如果是属性，则key为其所属分类，value为该属性、属性类别和个数。

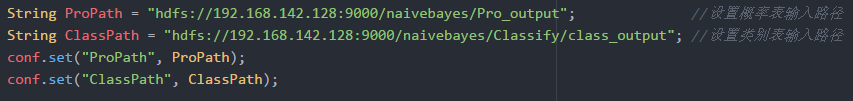


Reducer：先遍历获得总数，然后再用每一个value除以总数以获得概率，最后获得一张概率表

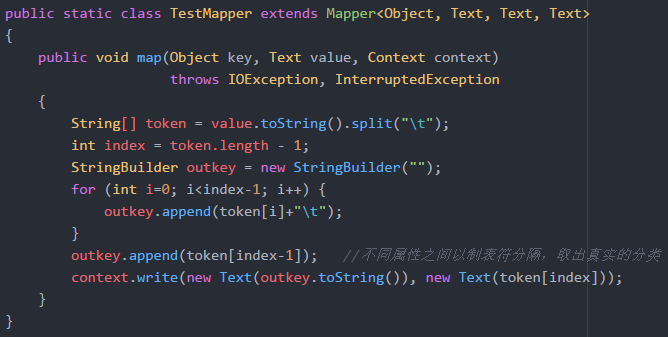


* NaiveBayesTest.java

Main：需要在main函数内设置类别表和概率表的路径，以供Reducer读取

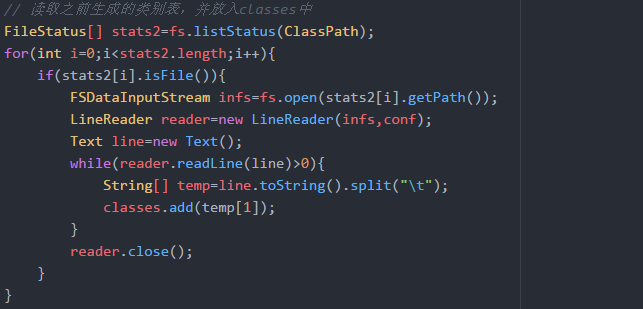


Mapper：将真实分类和属性分开，真实分类可以用于验证预测结果是否正确，属性则用于产生预测结果，key为全部属性，value为真实分类。

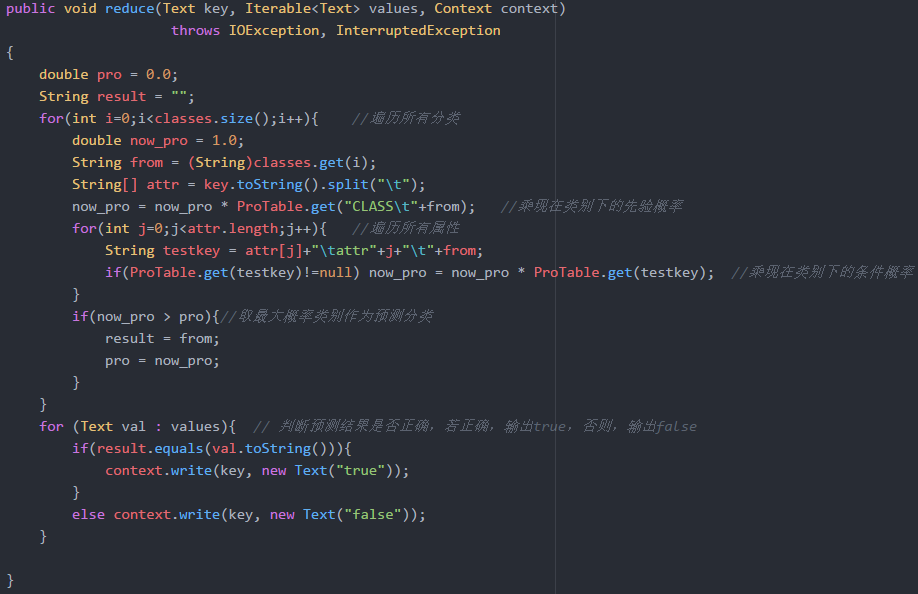


Reducer：首先在setup函数中读取类别表和概率表，将类别表放入数组，概率表放入map





然后在reduce函数中计算出该记录在所有分类下的概率，取概率最大的分类作为预测分类，与真实分类相比较，若两者相同，则即为”true”，否则记为”false”。



* NaiveBayesCount.java：一个简单的单词统计，输出预测结果中”true”和”false”的个数，因为比较简单，此处不再详述。

1. 实现难点：

实验的主要难点在于概率表的生成，为了生成概率表，需要总记录个数、各分类的个数，以及每个分类下各个属性的个数，先验概率和条件概率的计算如下：

先验概率 = 各分类的个数 / 总记录个数

条件概率 = 该分类下各个属性的个数 / 该分类的个数

各分类的个数，以及每个分类下各个属性的个数由NaiveBayesClass.java和NaiveBayesAttribute.java可以得到，而在NaiveBayesPro.java中，

将所有分类都分到同一个reducer中， 将所有分类的个数相加即可得到总记录个数，再将每个分类的个数除以总记录个数即可得到该分类的先验概率；

将同一个分类下的所有属性及其个数都分到同一个reducer中，将所有分类的个数相加即可得到该分类的个数，再将该分类下各个属性的个数除以该分类的个数，即可得到该分类下各个属性的条件概率。

1. 关键技术点：

实验的关键技术点在于读取之前生成的概率表，一开始并不知道怎样使程序自己读取HDFS文件系统上的文件，后来通过查找资料，发现只要设置好文件的路径，然后运用Path、FileSystem、FileStatus、FSDataInputStream、LineReader等类即可实现读取HDFS文件系统上的文件，从而实现了读取概率表操作，才能进行概率的计算。

1. **实验结果**
2. 编译所有java文件，并打包成jar包

|  |
| --- |
| javac NaiveBayesClass.java  jar –cvf NaiveBayesClass.jar NaiveBayesClass\*.class  javac NaiveBayesAttribute.java  jar –cvf NaiveBayesAttribute .jar NaiveBayesAttribute\*.class  javac NaiveBayesPro.java  jar –cvf NaiveBayesPro.jar NaiveBayesPro\*.class  javac NaiveBayesTest.java  jar –cvf NaiveBayesTest.jar NaiveBayesTest\*.class  javac NaiveBayesCount.java  jar –cvf NaiveBayesCount.jar NaiveBayesCount\*.class |

1. 启动Hadoop

|  |
| --- |
| /usr/local/Hadoop/sbin/start-all.sh |

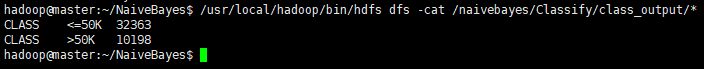
1. 准备相关文件夹和文件

|  |
| --- |
| /usr/local/hadoop/bin/hdfs dfs -mkdir /naivebayes  /usr/local/hadoop/bin/hdfs dfs -put data/adult\_data.txt data/adult\_test.txt /naivebayes/Data |

1. 提交NaiveBayesClass.jar，查看输出

|  |
| --- |
| /usr/local/hadoop/bin/hadoop jar NaiveBayesClass/NaiveBayesClass.jar NaiveBayesClass /naivebayes/Data/adult\_data.txt /naivebayes/Classify/class\_output  /usr/local/hadoop/bin/hdfs dfs -cat /naivebayes/Classify/class\_output/\* |

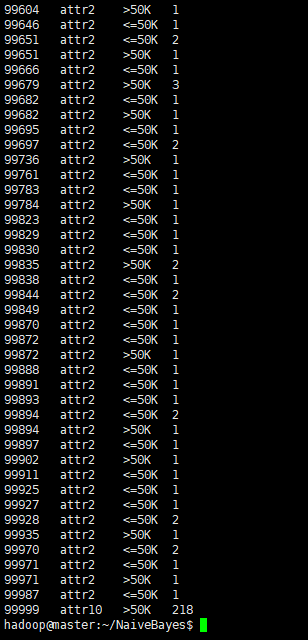
输出如下：



1. 提交NaiveBayesAttribute.jar，查看输出

|  |
| --- |
| /usr/local/hadoop/bin/hadoop jar NaiveBayesAttribute/NaiveBayesAttribute.jar NaiveBayesAttribute /naivebayes/Data/adult\_data.txt /naivebayes/Classify/attr\_output  /usr/local/hadoop/bin/hdfs dfs -cat /naivebayes/Classify/attr\_output/\* |

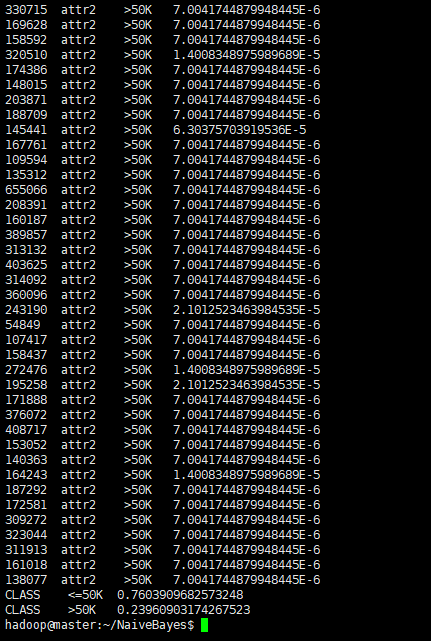
输出如下(部分)：



1. 提交NaiveBayesPro.jar，查看输出

|  |
| --- |
| /usr/local/hadoop/bin/hadoop jar NaiveBayesPro/NaiveBayesPro.jar NaiveBayesPro /naivebayes/Classify/\* /naivebayes/Pro\_output  /usr/local/hadoop/bin/hdfs dfs -cat /naivebayes/Pro\_output/\* |

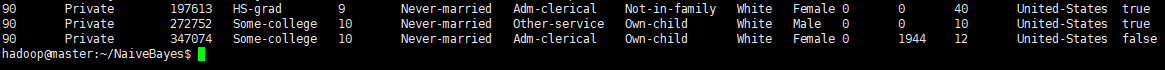
输出如下(部分)：



1. 提交NaiveBayesTest.jar，查看输出

|  |
| --- |
| /usr/local/hadoop/bin/hadoop jar NaiveBayesTest/NaiveBayesTest.jar NaiveBayesTest /naivebayes/Data/adult\_test.txt /naivebayes/Test\_output  /usr/local/hadoop/bin/hdfs dfs -cat /naivebayes/Test\_output/\* |

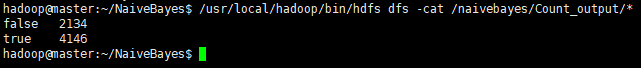
输出如下(部分)：



1. 提交NaiveBayesCount.jar，查看输出

|  |
| --- |
| /usr/local/hadoop/bin/hadoop jar NaiveBayesCount/NaiveBayesCount.jar NaiveBayesCount /naivebayes/Test\_output/\* /naivebayes/Count\_output  /usr/local/hadoop/bin/hdfs dfs -cat /naivebayes/Count\_output/\* |

输出如下：



可以看到，在本次实验中，朴素贝叶斯算法的正确率为66%

1. **总结**
2. 这次实验在理解算法上面并不难，时间主要花费在MapReduce模型的构建和优化上。我们的代码一共经历了三个版本的迭代：
3. 第一个版本完全是根据此次的数据量身打造出来的，将分类统计和属性统计放在一起，生成概率表时只考虑两种情况，不具备普适性，同时在验证预测结果的正确性上，使用了两个java文件去实现验证，验证前还需要将测试集和预测结果手动放在同一个文件夹下，步骤比较繁琐；
4. 第二个版本增强了普适性的要求，将分类统计和属性统计分开，生成概率表时遍历类别表，而不是只考虑两种情况，同时将一些判断条件进行修改，使之能够适用于其他数据；
5. 第三个版本进一步考虑属性中不同属性取值相同的问题，比如年龄这一属性有一个取值为10，而受教育年限也有一个取值为10，当进行统计时，这两个会被分到同一个reducer中，造成错误，所以我们在属性后加上该属性为第几个属性，即年龄为变成”10 attr0”，而受教育年限会变成”10 attr4”，这样两者就可以区分了；同时对验证进行优化，在预测出结果的同时判断结果是否正确，这样既可以减少一个java文件，也不用手动移动文件，简化了步骤；
6. 一开始时由于不熟悉hadoop一些相关的api，所以每次重新跑代码时都需要手动删除已有的文件夹，十分麻烦，后来使用filesystem判断文件夹是否已存在，若存在则自动删除该文件夹；还有自动读取HDFS文件系统上的文件的问题，通过查找资料，运用Path、FileSystem、FileStatus、FSDataInputStream、LineReader等类中的相关方法解决了问题；同时相关的api有些已经改变，比如创建job不再使用new Job()方法，而是使用Job.getInstance()方法创建job；
7. 在使用数据时，发现网上下载的数据每一行最后有一个英文句号’.’，所以原始数据中的分类为’>50K.’和’<=50K.’，而通过MapReduce之后输出的分类为’>50K’和’<=50K’，没有最后的句号，一开始没有发现这个问题，等到验证时发现预测结果全错了，检查很久后才发现这个问题，为了方便分割数据和验证，我们对原始数据进行了处理，使得属性之间以及属性和分类之间使用制表符分隔，去掉原始数据中分类中的英文句号；
8. 在实现了hadoop版本的朴素贝叶斯算法后，我们又写了一个spark版本的程序，但是由于scala的配置和spark相关库的引用出现了一定问题，研究了很久也没有解决这个问题，所以导致我们写出来之后不能进行相关的编译，也就没有办法进行运行测试，由于时间有限，这个只能留待日后进行实现；
9. 关于后续改进：本次的实验结果还不够完善，还可进一步优化，比如对一些属性进行分段统计，如对年龄进行分段，而不是直接使用某一个年龄数；还可以对各属性进行权重分配，因为每个属性对预测结果的影响不一定相同，有可能学历比婚姻状况对结果的影响更大，所以两者的权值应该不同；还可以对数值类型的属性进行平滑处理，而不是将其当作文本类型的数据进行处理；还可以加上决策树…等等；

**附录：**

·小组分工：

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 实现部分 | 贡献率 |
| 15352352 | 肖楚铭 | 关于训练集的处理和生成概率表的java代码部分 | 40% |
| 15352354 | 肖想 | 对测试集进行预测的java代码部分 | 30% |
| 15352351 | 项乐彬 | 对结果进行统计的java代码部分 | 30% |

·参考书籍：《数据算法——Hadoop/Spark大数据处理技巧》 作者：Mahmoud Parsian

补充：

1. 以上内容提到部分必须包括，如需其他部分作说明可自行补充
2. 如果是实现的某一篇论文的算法，提交内容里面需要有相应的论文资料
3. 提交的格式为：组长学号\_组员1学号\_组员2学号.zip，其中需要包括①代码文件夹code，②实验报告文档，以组长学号\_组员1学号\_组员2学号.pdf命名，③readme文档：对代码的运行条件和代码的解释等。