**目录**

[一：贝叶斯分类器理论介绍 1](#_Toc15299)

[二：贝叶斯分类器训练的MapReduce算法设计 2](#_Toc7524)

[三：源代码清单 3](#_Toc18548)

[四：数据集说明 12](#_Toc29467)

[五：程序运行说明 13](#_Toc23697)

[六：实验结果分析 17](#_Toc23833)

[七：源文件 17](#_Toc31026)

**一：贝叶斯分类器理论介绍**

在[机器学习](https://zh.wikipedia.org/wiki/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0" \o "机器学习)中，朴素贝叶斯分类器是一系列以假设特征之间强（朴素）[独立](https://zh.wikipedia.org/wiki/%E7%B5%B1%E8%A8%88%E7%8D%A8%E7%AB%8B%E6%80%A7" \o "统计独立性)下运用[贝叶斯定理](https://zh.wikipedia.org/wiki/%E8%B4%9D%E5%8F%B6%E6%96%AF%E5%AE%9A%E7%90%86" \o "贝叶斯定理)为基础的简单[概率分类器](https://zh.wikipedia.org/w/index.php?title=%E6%A6%82%E7%8E%87%E5%88%86%E7%B1%BB%E6%B3%95&action=edit&redlink=1)。

朴素贝叶斯自20世纪50年代已广泛研究。在20世纪60年代初就以另外一个名称引入到[文本信息检索](https://zh.wikipedia.org/wiki/%E6%96%87%E6%9C%AC%E4%BF%A1%E6%81%AF%E6%A3%80%E7%B4%A2" \o "文本信息检索)界中， 并仍然是[文本分类](https://zh.wikipedia.org/w/index.php?title=%E6%96%87%E6%A1%A3%E5%88%86%E7%B1%BB&action=edit&redlink=1)的一种热门（基准）方法，文本分类是以[词频](https://zh.wikipedia.org/w/index.php?title=%E8%AF%8D%E8%A2%8B&action=edit&redlink=1)为特征判断文件所属类别或其他（如[垃圾邮件](https://zh.wikipedia.org/wiki/%E5%8F%8D%E5%9E%83%E5%9C%BE%E9%82%AE%E4%BB%B6%E6%8A%80%E6%9C%AF" \o "反垃圾邮件技术)、合法性、体育或政治等等）的问题。通过适当的预处理，它可以与这个领域更先进的方法（包括[支持向量机](https://zh.wikipedia.org/wiki/%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%9C%BA" \o "支持向量机)）相竞争。 它在自动[医疗诊断](https://zh.wikipedia.org/wiki/%E8%AF%8A%E6%96%AD" \o "诊断)中也有应用。

理论上，概率模型分类器是一个条件概率模型。

（1）朴素贝叶斯分类器的条件概率为：



其中：

：表示测试集文件中的单词

：表示训练集中的类

：表示测试集文件中的单词t在训练集中类c里出现的总次数

：表示训练集中类c里的单词总数

：整个训练集文档V中的不同单词总数

条件概率模型做了平滑处理，防止测试集文件中的单词t在训练集中类c里没有出现的情况。

（2）朴素贝叶斯分类器的先验概率为：



：表示训练集中类c里的单词总数

：表示训练集文档V中的单词总数

这是基于单词的先验概率，是在基于文件的先验概率上的优化，可以减少整个训练集文档中有很多类别，但类别中只有很少的文件的情况的影响。

（3）根据朴素贝叶斯分类器的独立性假设的前提，可得到如下分类规则：

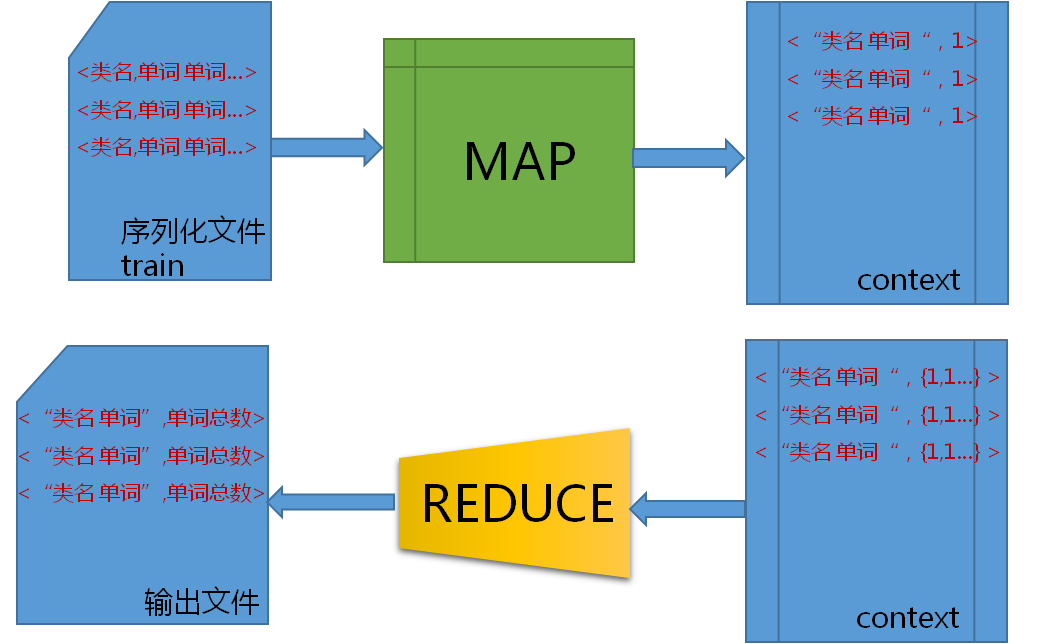


对训练集中类的集合分别计算上述数值，取最大的值的那个类就是预测的结果。

**二：贝叶斯分类器训练的MapReduce算法设计**

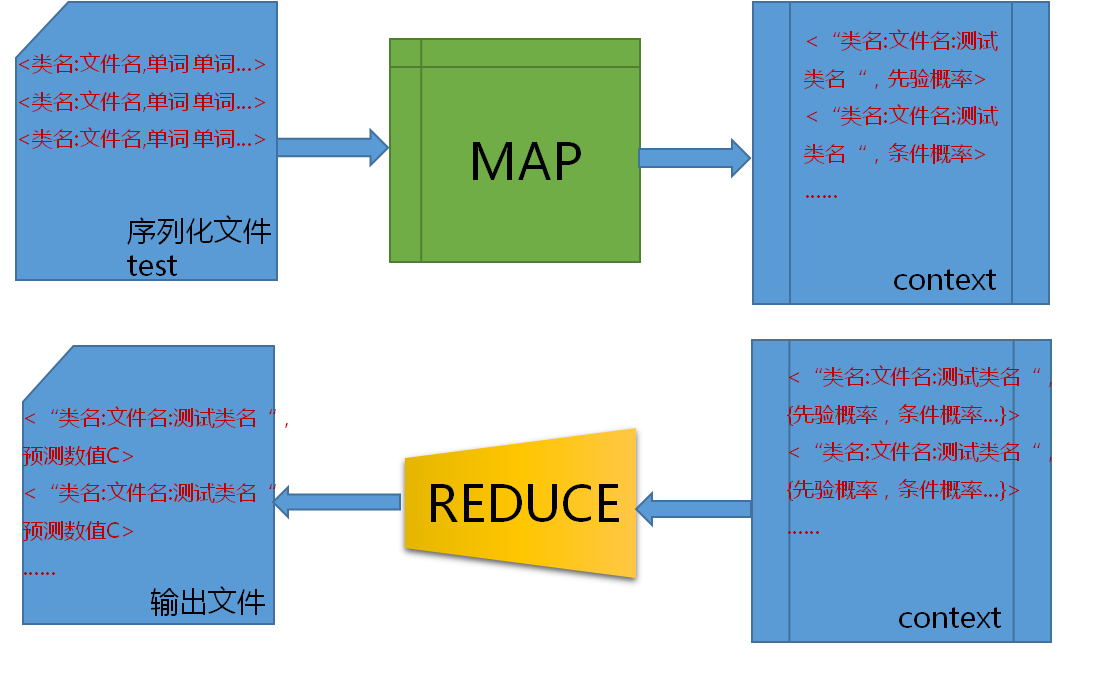
整个工程包含CountryTrain.java、Prediction.java和Evaluation.java三个程序，其中CountryTrain.java、Prediction.java是MapReduce程序，Evaluation.java是单机java程序。

CountryTrain.java程序中MapReduce作业输入文件是序列化文件train，通过对训练集进行序列化操作得到的，其格式为一个一个的Key-Value对：<类名，单词 单词 ......>，这也是Map的输入。通过Map后输出的Key为（类名 单词）、Value为 （1），其中的含义是该单词在该类出现了1次。通过Combine和Reduce后输出的Key为（类名 单词）、Value为（单词总数），其含义为该单词在该类出现的总次数。



CountryTrain.java程序中MapReduce的Data Flow图

Prediction.java程序中MapReduce作业输入文件是序列化文件test，通过对测试集进行序列化操作得到的，其格式为一个一个的Key-Value对：<类名:文件名，单词 单词 ......>，这也是Map的输入。通过Map后输出的Key为（类名:文件名:测试的类名）、Value为 （先验概率或条件概率），其中的含义是该类下的测试文件属于这个测试类的先验概率和条件概率。通过Combine和Reduce后输出的Key为（类名:文件名:测试的类名）、Value为（预测数值），其含义为该类下的测试文件属于这个测试类的预测数值。



Prediction.java程序中MapReduce的Data Flow图

Evaluation.java程序是单机java程序，该程序根据Prediction.java程序中MapReduce的输出，求得某类下的测试文件属于哪个测试类，获得key为（真实类名:文件名）,value为（预测类名)的哈希表。并根据此哈希表求得每个类下的Precision，Recall和F1值，最后求得Precision，Recall和F1的平均值。

**三：源代码清单**

**SequenceFileWriter.java**

import java.net.URI;

import java.io.File;

import java.io.FileReader;

import java.io.BufferedReader;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.SequenceFile;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.IOUtils;

public class SequenceFileWriter {

private static String file2String(File file) throws IOException{

BufferedReader reader = new BufferedReader(new FileReader(file));

String line = null;

String result = "";

while((line = reader.readLine()) != null){

if(line.matches("[a-zA-Z]+")){ //过滤掉以数字开头或无意义的词，只选择英文单词

result += line + " ";

}

}

reader.close();

return result;

}

//传入两个参数，args[0]为所读取的本地文件夹路径，而非HDFS文件路径;args[1]为想要写入文件的路径

public static void main(String[] args) throws IOException {

File[] dirs = new File(args[0]).listFiles();

// 为创建SequenceFile.Writer准备参数

String uri = args[1];

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(URI.create(uri), conf);

Path path = new Path(uri);

Text key = new Text();

Text value = new Text();

SequenceFile.Writer writer = null;

try {

writer = SequenceFile.createWriter(fs, conf, path, key.getClass(),

value.getClass());

for(File dir: dirs) {

File[] files = dir.listFiles();

for(File file: files) {

key.set(dir.getName()); // Key为类名,是CountryTrain数据集序列化的格式 //key.set(dir.getName() + ":" + file.getName()); //key为类名+":"+文件名，是CountryTest数据集序列化的格式

value.set(file2String(file)); // value为文件内容，是单词串

writer.append(key, value);

}

}

} finally {

IOUtils.closeStream(writer);

}

}

}

**Prediction.java**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

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.input.SequenceFileInputFormat;

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

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class CountryTrain {

//读取train文件,输入格式为：类名 单词 单词 ......

public static class CountryTrainMap extends Mapper<Text, Text, Text, IntWritable> {

private final IntWritable one = new IntWritable(1);

private Text class\_word = new Text();

public void map(Text key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

StringTokenizer token = new StringTokenizer(line);

while (token.hasMoreTokens()) {

class\_word.set(key +" "+ token.nextToken());

context.write(class\_word,one);

}

}

}

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

public 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)); //最后输出<<类名，单词>,单词总数>

System.out.println(key + "\t" + sum);

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

Job job = new Job(conf);

job.setJarByClass(CountryTrain.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(CountryTrainMap.class);

job.setReducerClass(CountryTrainReduce.class);

job.setInputFormatClass(SequenceFileInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

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

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

job.waitForCompletion(true);

}

}

**Prediction.java**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Enumeration;

import java.util.Hashtable;

import java.util.StringTokenizer;

import java.io.File;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.filecache.DistributedCache;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.\*;

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.input.SequenceFileInputFormat;

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

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.Tool;

@SuppressWarnings("deprecation")

public class Prediction extends Configured implements Tool{

//读取test文件,输入格式为：真实类名:文件名 单词 单词 ......

public static class PredictionMap extends Mapper<Text, Text, Text, DoubleWritable> {

private static Hashtable<String,Integer> toTct (File file) throws IOException{ //函数toTct获得(key:真实类名 单词,value:该单词在该类出现的次数)的哈希表

Hashtable<String,Integer> ht = new Hashtable<String,Integer>();

BufferedReader reader = new BufferedReader(new FileReader(file));

Integer value;

String line = null;

String key = "";

while((line = reader.readLine()) != null){

StringTokenizer token = new StringTokenizer(line);

key=token.nextToken()+" "+token.nextToken();

value=Integer.parseInt(token.nextToken());

ht.put(key, value);

}

reader.close();

return ht;

}

private static Hashtable<String,Integer> toTtc (File file) throws IOException{ //函数toTtc获得(key:真实类名,value:该类中单词数)的哈希表

Hashtable<String,Integer> ht = new Hashtable<String,Integer>();

BufferedReader reader = new BufferedReader(new FileReader(file));

Integer num;

Integer sum=0;

String line = null;

String classname = null;

while((line = reader.readLine()) != null){

StringTokenizer token = new StringTokenizer(line);

classname=token.nextToken();

token.nextToken();

num=Integer.parseInt(token.nextToken());

if(ht.containsKey(classname)){

sum=ht.get(classname)+num;

ht.put(classname, sum);

}

else{

ht.put(classname, num);

}

}

reader.close();

return ht;

}

private static Integer toB (File file) throws IOException{ //函数toB获得整个文档中的不同单词数

Hashtable<String,Integer> ht = new Hashtable<String,Integer>();

BufferedReader reader = new BufferedReader(new FileReader(file));

Integer sum=0;

String line = null;

String word = "";

while((line = reader.readLine()) != null){

StringTokenizer token = new StringTokenizer(line);

token.nextToken();

word=token.nextToken();

if(!ht.containsKey(word)){

ht.put(word, 1);

++sum;

}

}

reader.close();

return sum;

}

Hashtable<String,Integer> Tct=new Hashtable<String,Integer>();

Hashtable<String,Integer> Ttc=new Hashtable<String,Integer>();

Integer B=0;

Double Ttv=0.00000; //Ttv:整个文档中的单词数

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

Path[] paths = DistributedCache.getLocalCacheFiles(context.getConfiguration());

if (paths != null && paths.length > 0) {

File file=new File(paths[0].toString());

try {

Tct=toTct(file);

Ttc=toTtc(file);

B=toB(file);

Enumeration<Integer> e = Ttc.elements();

while( e.hasMoreElements() ){

Ttv+=e.nextElement();

}

}

catch (Exception e )

{

e.printStackTrace();

}

}

}

private final DoubleWritable probability = new DoubleWritable();

private Text word = new Text();

private Text index = new Text();

private Text classname = new Text();

private Text keys = new Text();

public void map(Text key, Text value, Context context) throws IOException, InterruptedException {

Enumeration<String> ee = Ttc.keys();

while( ee.hasMoreElements() ){

classname.set(ee.nextElement());

keys.set(key+":"+classname); //map输出的key的格式为：类名:文件名:测试的类名

String line = value.toString();

StringTokenizer token = new StringTokenizer(line);

while (token.hasMoreTokens()) {

word.set(token.nextToken());

index.set(classname+" "+word);

if(Tct.containsKey(index.toString())){

probability.set((Tct.get(index.toString())+1.0)/(Ttc.get(classname.toString())+B));

context.write(keys,probability); //测试集中单词存在该类中的条件概率

}

else{

probability.set((1.0)/(Ttc.get(classname.toString())+B));

context.write(keys,probability); //测试集中单词不存在该类中的条件概率（平滑处理）

}

}

probability.set(Ttc.get(classname.toString())/Ttv); //先验概率

context.write(keys,probability);

}

}

}

public static class PredictionReduce extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

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

double C = 0.0;

for (DoubleWritable val : values) {

C += Math.log(val.get());

}

context.write(key,new DoubleWritable(C)); //最后输出<真实类名:文件名:测试的类名 ,概率>

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

DistributedCache.addCacheFile(new Path("hdfs://hadoop:9000/user/root/CountryTrain/output/part-r-00000.txt").toUri(),conf);

Job job = new Job(conf);

job.setJarByClass(Prediction.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(DoubleWritable.class);

job.setMapperClass(PredictionMap.class);

job.setReducerClass(PredictionReduce.class);

job.setInputFormatClass(SequenceFileInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

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

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

job.waitForCompletion(true);

}

@Override

public int run(String[] arg0) throws Exception {

// TODO Auto-generated method stub

return 0;

}

}

**Evaluation.java**

import java.io.BufferedReader;

import java.io.File;

import java.io.FileReader;

import java.io.IOException;

import java.util.Enumeration;

import java.util.Hashtable;

import java.util.List;

import java.util.StringTokenizer;

import java.util.Vector;

public class Evaluation {

static String[] ClassSet={ //根据函数toTtc获得类名集合 "ISRAEL","CANA","MEX","EEC","FRA","UK","HKONG","JAP","ARG","GFR","BELG","SAFR","BRAZ","ALB","MALAY","ITALY","INDON","POL","AUSTR","SKOREA","CHINA","SINGP","RUSS","THAIL","INDIA","NETH","CZREP","PHLNS","FIN"};

static int ClassSum=ClassSet.length; //类的总数

private static Hashtable<String,String> predictionClass(File file) throws IOException{ //函数predictionClass获得(key:真实类名:文件名,value:预测类名)的哈希表

Hashtable<String,String> ht = new Hashtable<String,String>();

Hashtable<String,Double> cache = new Hashtable<String,Double>();

BufferedReader reader = new BufferedReader(new FileReader(file));

Double value;

Double max=-Double.MAX\_VALUE;

String line = null;

String key = null;

String a = null;

Integer index=ClassSum;

while((line = reader.readLine()) != null&&(index--) != 0){

StringTokenizer token = new StringTokenizer(line);

key=token.nextToken();

value=Double.parseDouble(token.nextToken());

if(value>max){

max=value;

}

cache.put(key, value);

if(index==0) {

Enumeration<String> e = cache.keys();

while( e.hasMoreElements() ){

a=e.nextElement();

if(max==cache.get(a)){

String[] part =a.split(":");

ht.put(part[0]+":"+part[1],part[2]);

}

}

index=ClassSum;

max=-Double.MAX\_VALUE;

cache.clear();

}

}

reader.close();

return ht;

}

public static void main(String[] args) throws Exception {

String a=null;

String b=null;

int index=0,f;

double TP=0.0,FN=0.0,FP=0.0,TN=0.0;

double P,R,F;

double Precision[]=new double[ClassSum];

double Recall[]=new double[ClassSum];

double F1[]=new double[ClassSum];

double sumuP=0.0,sumR=0.0,sumF=0.0;

Hashtable<String,String> pc = new Hashtable<String,String>();

File file=new File(args[0].toString());

pc=predictionClass(file);

for(String c: ClassSet){ //遍历类集合

Enumeration<String> e = pc.keys();

while( e.hasMoreElements() ){

a=e.nextElement();

String[] part =a.split(":"); //part[0]是真实类名

b=pc.get(a); //b是预测类名

if((c.equals(part[0]))&&c.equals(b)) TP++;

else if(c.equals(part[0])&&!c.equals(b)) FN++;

else if(!c.equals(part[0])&&c.equals(b)) FP++;

else if(!c.equals(part[0])&&!c.equals(b)) TN++;

}

P=TP/(TP+FP); //Precision

R=TP/(TP+FN); //Recall

F=2\*P\*R/(P+R); //F1

f=index++;

Precision[f]=P;

Recall[f]=R;

F1[f]=F;

TP=0.0;FN=0.0;FP=0.0;TN=0.0;

}

for(index=0;index<ClassSum;index++){

sumuP+=Precision[index];

sumR+=Recall[index];

sumF+=F1[index];

}

System.out.println("平均Precision精度值："+sumuP/ClassSum);

System.out.println("平均Recall精度值："+sumR/ClassSum);

System.out.println("平均的调和均值："+sumF/ClassSum);

}

}

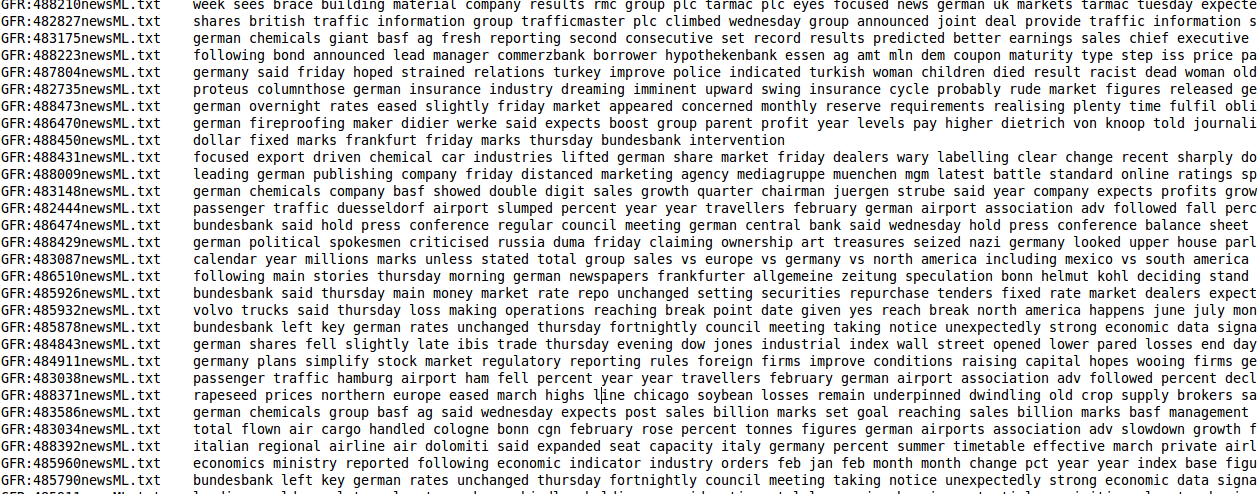
**四：数据集说明**

工程选择Country文件做数据集，舍去类别中文档数少于80的类别，剩下如下的类别：

{"ISRAEL","CANA","MEX","EEC","FRA","UK","HKONG","JAP","ARG","GFR","BELG","SAFR","BRAZ","ALB","MALAY","ITALY","INDON","POL","AUSTR","SKOREA","CHINA","SINGP","RUSS","THAIL","INDIA","NETH","CZREP","PHLNS","FIN"};

选取上述类别中3/4的文档数作为训练集，1/4的文档数作为测试集。分为CountryTrain和CountryTest文件。

运行SequenceFileWriter.java分别对CountryTrain和CountryTest文件做序列化操作，得到train和test序列化文件。其中train的格式为：<真实类名，单词 单词 ....>的Key-Value对；test的格式为：<真实类名：文件名，单词 单词 ....>的Key-Value对。

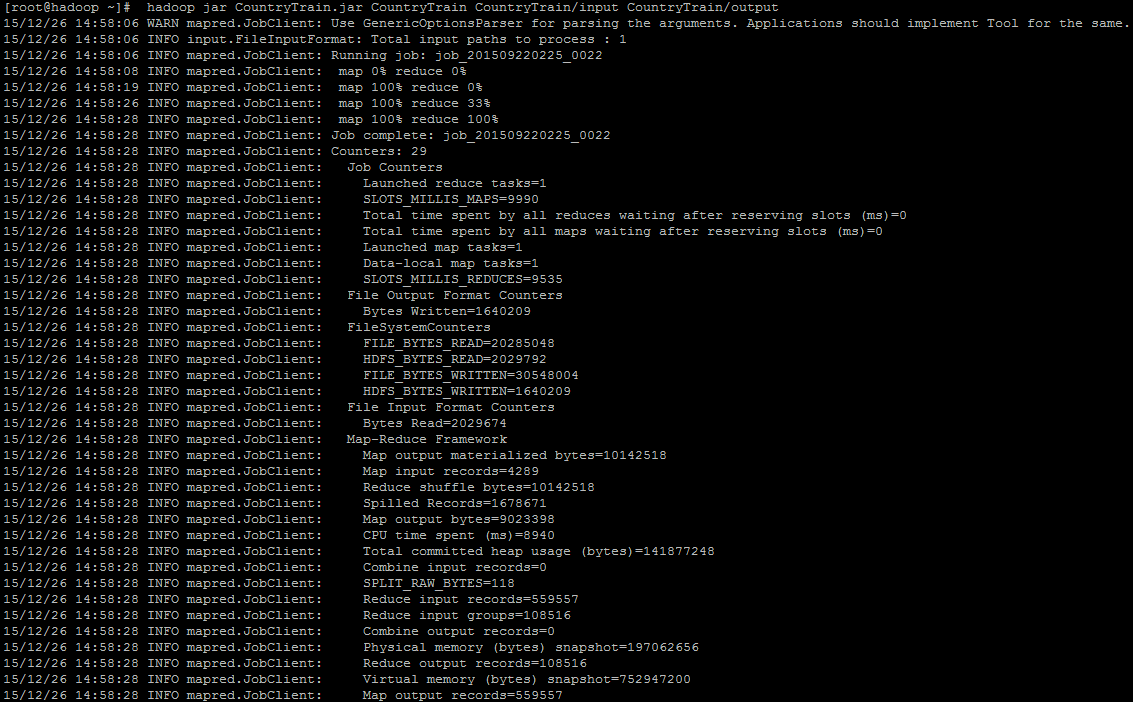


test序列化文件的内容图

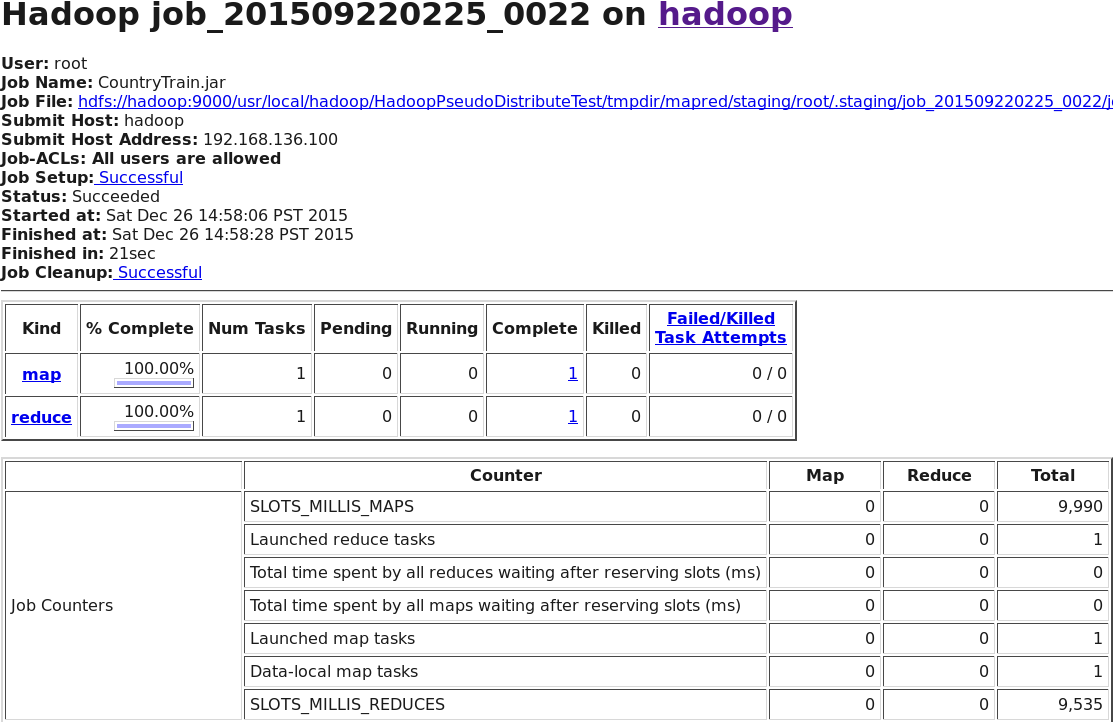
**五：程序运行说明**

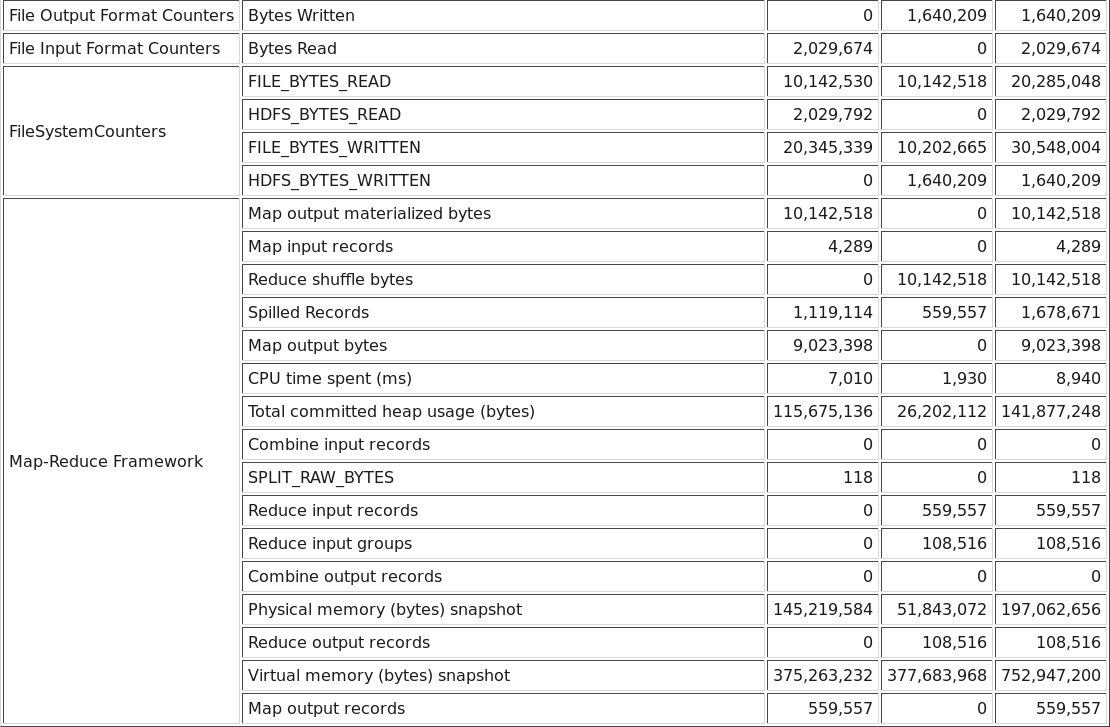
1. 运行CountryTrain.java

运行时有1个Map任务，1个Reduce任务



CountryTrain.java程序运行截图

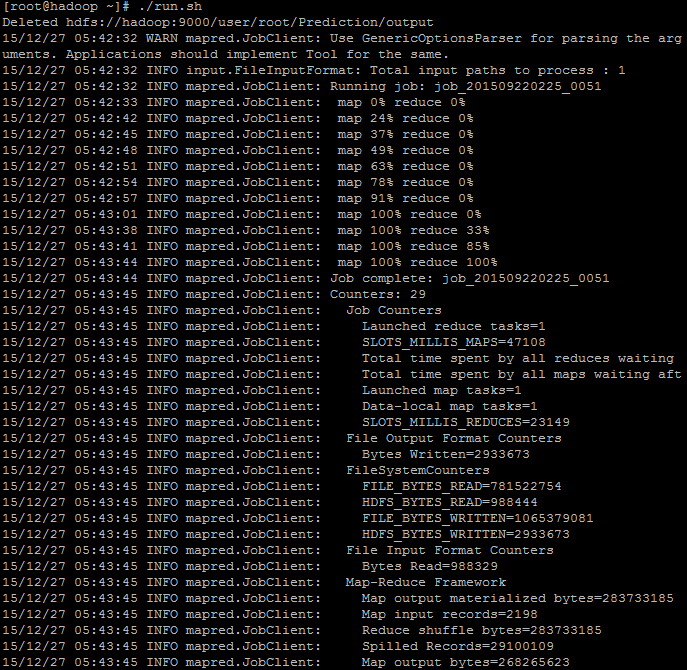


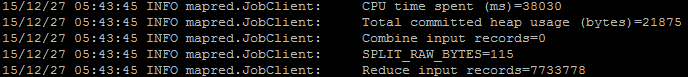


CountryTrain.java运行基于Web监控截图

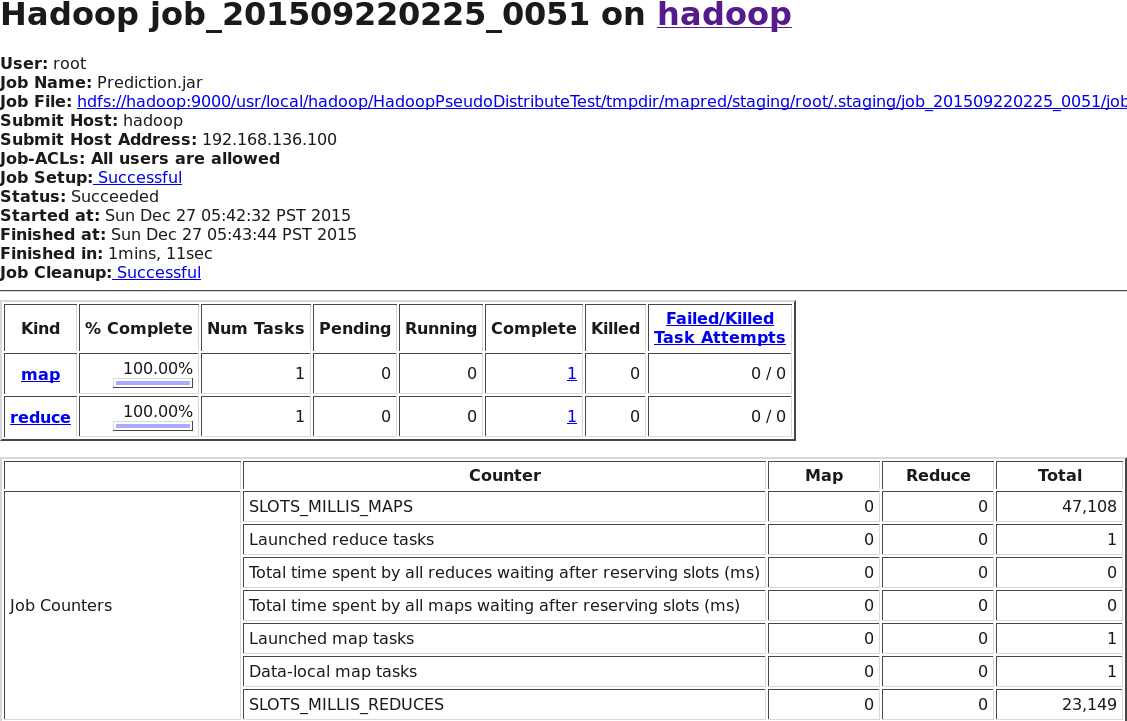
1. 运行Prediction.java

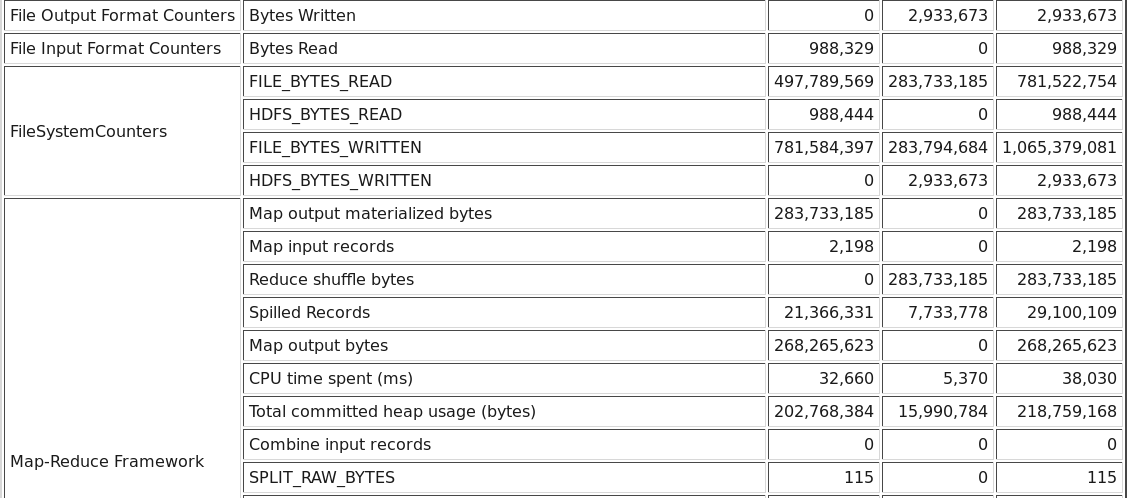
运行时有1个Map任务，1个Reduce任务





Prediction.java程序运行截图



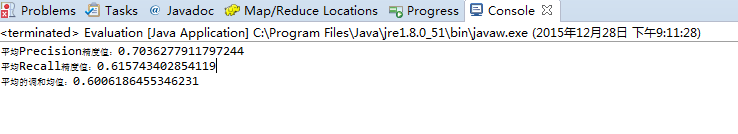




Prediction.java运行基于Web监控截图

1. 运行Evaluation.java

由于是单机Java程序，没有Map任务和Reduce任务。

Evaluation.java程序运行截图

**六：实验结果分析**

最后，求得分类结果的Precision，Recall和F1值分别为：

Precision=0.7036277911797244

Recall=0.615743402854119

F1=0.6006186455346231

实验结果达到了约60%的准确率，说明朴素贝叶斯分类器的效果还是很不错的。使用这个朴素贝叶斯分类器可以很方便的对文档进行归类。

如果增加训练集的大小，预测的结果还会更好。

**七：源文件**

整个工程的源文件：https://github.com/ccyyx/NaiveBayesClassifier.git