云计算实践大作业

姓名：黄杨峻

学号：17341059

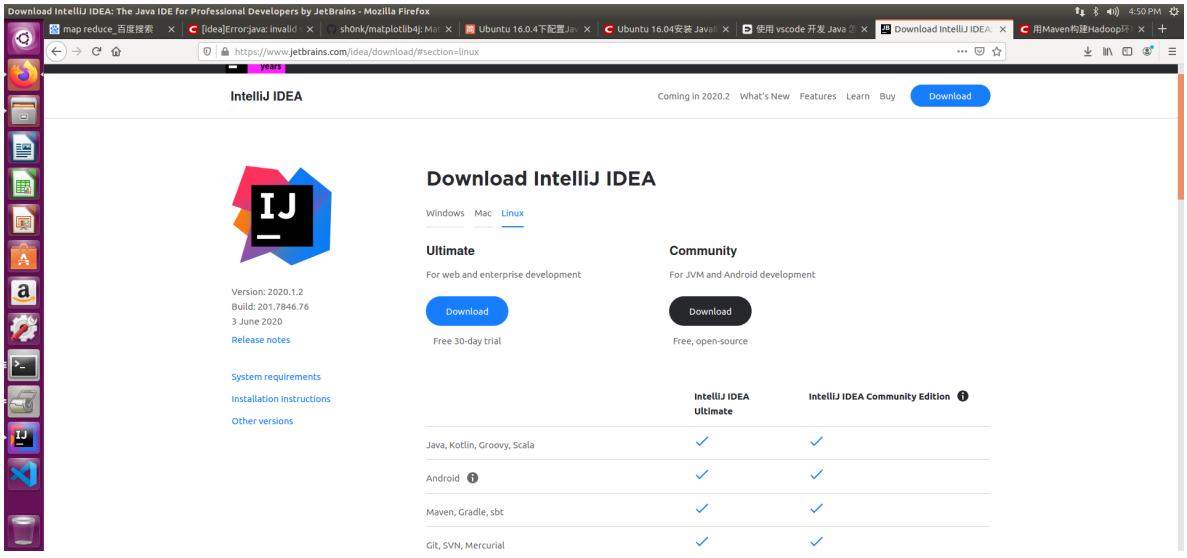
实验目标

用MapReduce实现EM算法（难度分4）

前期准备

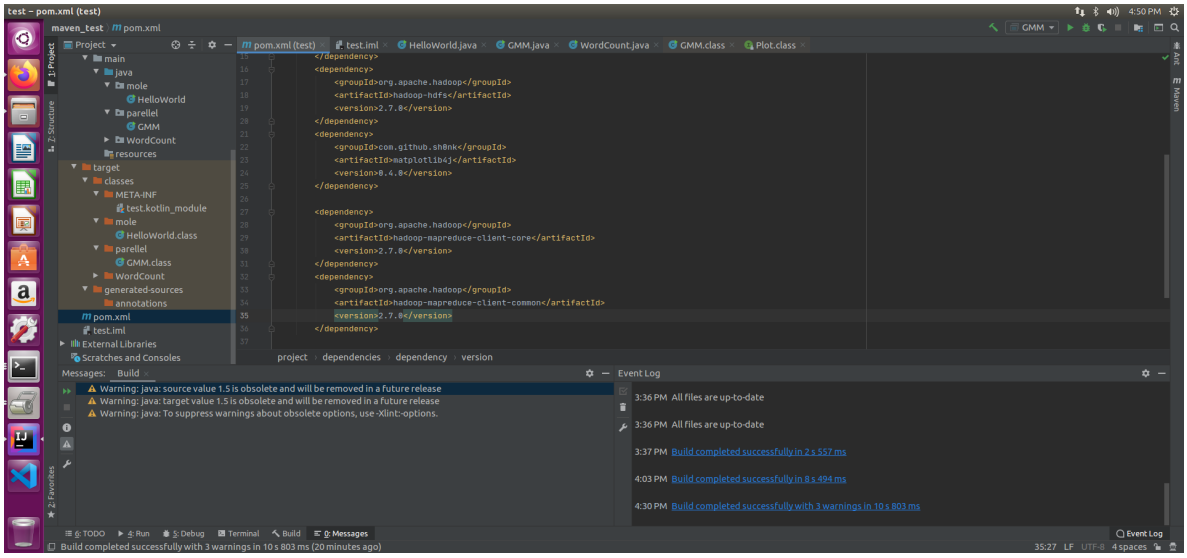
IDEA+Maven配置

IDEA是一款 JetBrains 全家桶中的JAVA IDE，我认为是目前最好用的JAVA编辑器，拥有强大的查错、代码补全、代码编译功能，是VSCODE、Eclipse不能比的。而Ubuntu版本的IDEA的下载也很方便，直接到官网下就可以。

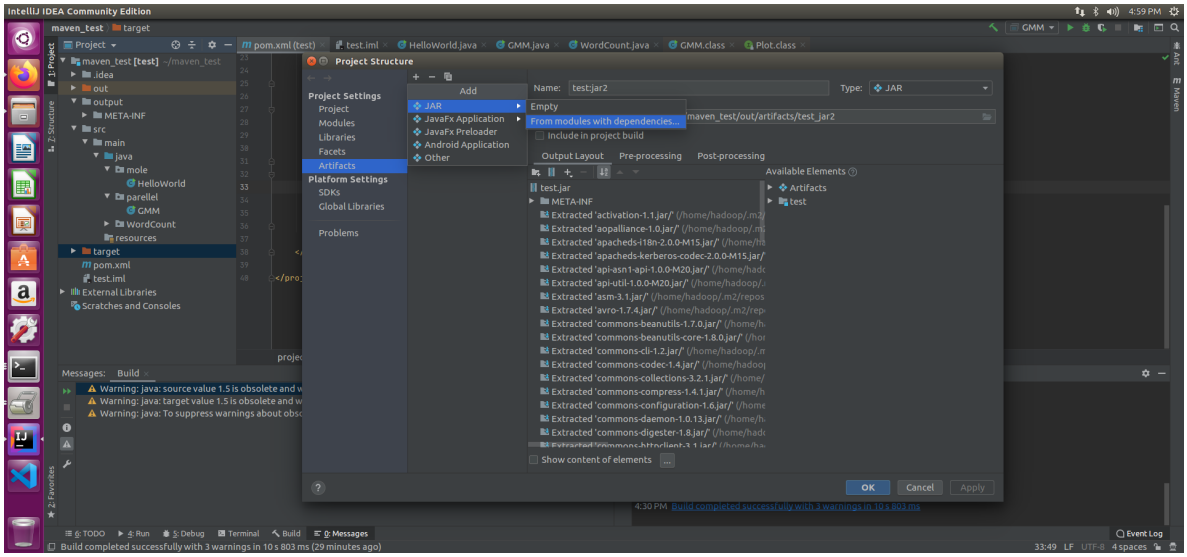


因为Ultimate版本要收钱，这里我用的是Community版本。

由于项目可能会用到比较多外面的包（包括科学计算库、Hadoop库、画图库之类的），我认为手动下代码到CLASSPATH太过麻烦，需要一种更高效的依赖管理以满足需求。所以这里我采用了Maven结构。Maven最大的优点就是依赖包的下载，只需在POM.xml中加入依赖项就能自动拉取，不用担心各种依赖缺失。



这里打包jar的操作也比较方便，首先点开File->Project Structure->Artifacts，然后添加build artifacts的规则，选from modules with dependencies、最后选好要打包的类（注意一定要包含main方法）就可以了。



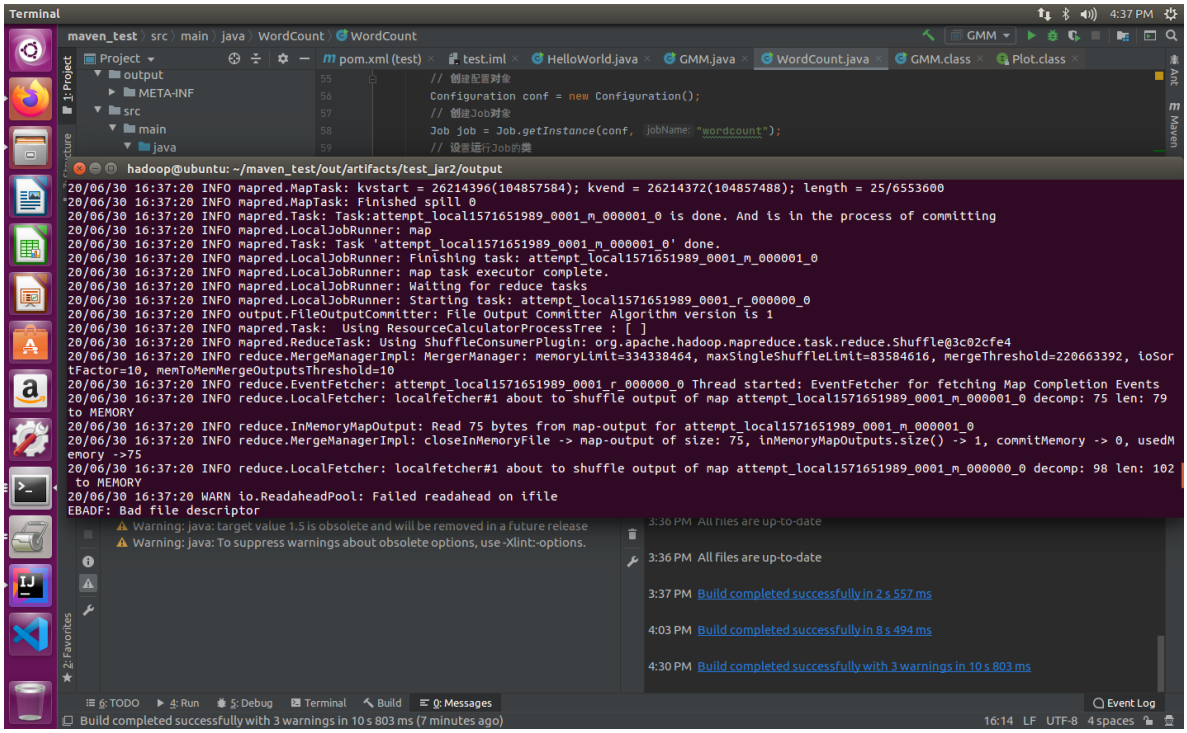
MapReduce测试

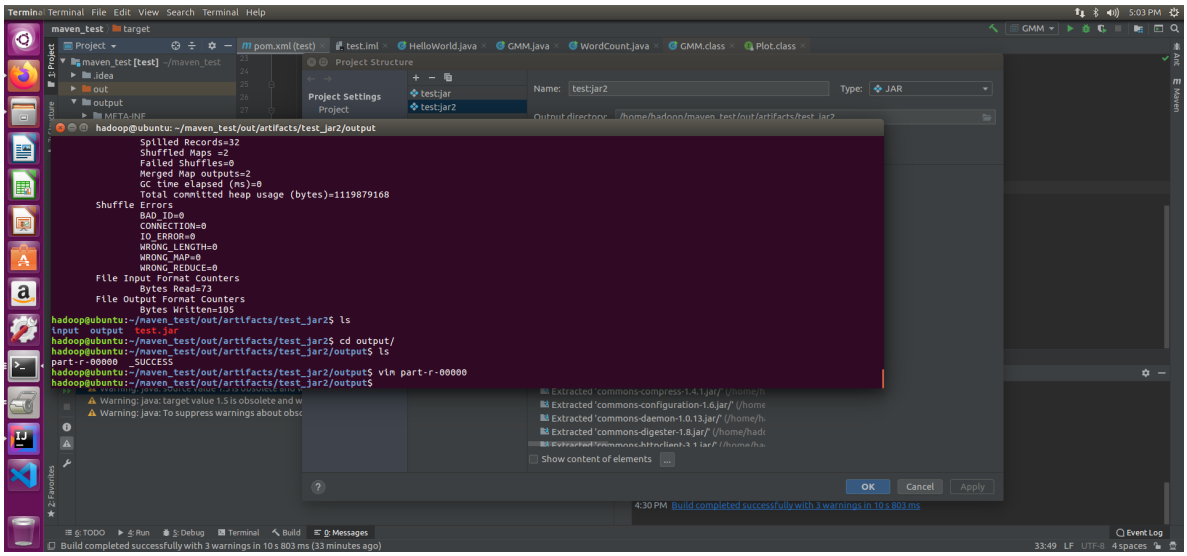
MapReduce是一个编程模型，用于处理和生成（key，value）对数据集合。在实验二中，我们用`wordCount.java`打包jar包，然后提交给MapReduce调度系统，调度系统将这些任务调度到集群中多台可用的机器上。

这里我继续用`wordCound.java`进行测试，打包成test.jar，在input文件夹中创建两个文本并填入部分单词，然后通过输入指令进行MapReduce

```
hadoop jar test.jar input output
```

结果





成功了~

理论

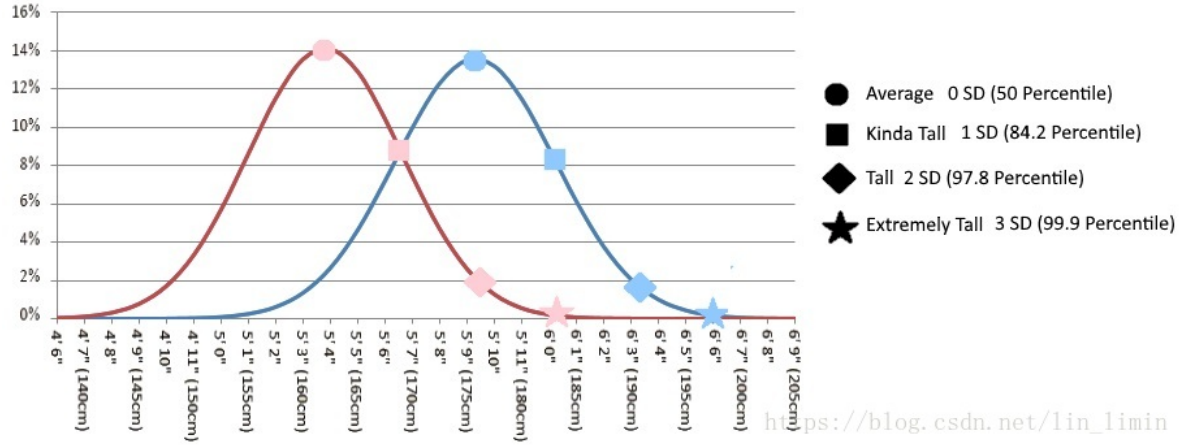
EM算法

谈到EM算法，我们首先要介绍高斯混合模型，众所周知，一维的高斯分布可以描述为

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp -\frac{(x-\mu)^2}{2\sigma^2}$$

其中 σ 为标准差，μ 为均值。而多维变量高斯分布的联合概率密度函数可以表示为

$$f(X) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}\exp[-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)]$$



给出一组任意形状的非线性函数，我们都能用若干个混合的一维或多维高斯模型概率密度函数的叠加曲线拟合出来，这个叠加的函数我们就把它称作GMM，它可以用下面这个公式来描述

$$p(y) = \sum_{k=1}^{K} \pi_k N(y|\mu_k, \Sigma_k)$$

其中π_k为选择第k个模型的概率，$N(y|\mu, \Sigma)$为高斯分布函数。

这里举一个GMM的例子，上图中展示的男女身高的概率分布模型，假设男女生的身高各自满足一个高斯分布，那么我们可以在模型中考虑用两个一维的高斯分布的叠加来进行建模。

在GMM的参数估计中，μ和Σ往往可以通过极大似然对数求导法得到，但π的参数估计不能通过极大似然法来求，因为我们预先不清楚每一个点到底属于哪个高斯函数。所以我们要引入EM算法。

EM算法可以用以下公式组来描述

Estep：

$$\gamma_{nk} = \frac{\pi_k N(X_n|\mu_k, \Sigma_k)}{\Sigma_{j=1}^{K} \pi_j N(X_n|\mu_j, \Sigma_j)}$$

其中$N(x|\mu, \Sigma)$为高斯分布的概率密度函数。

MStep:

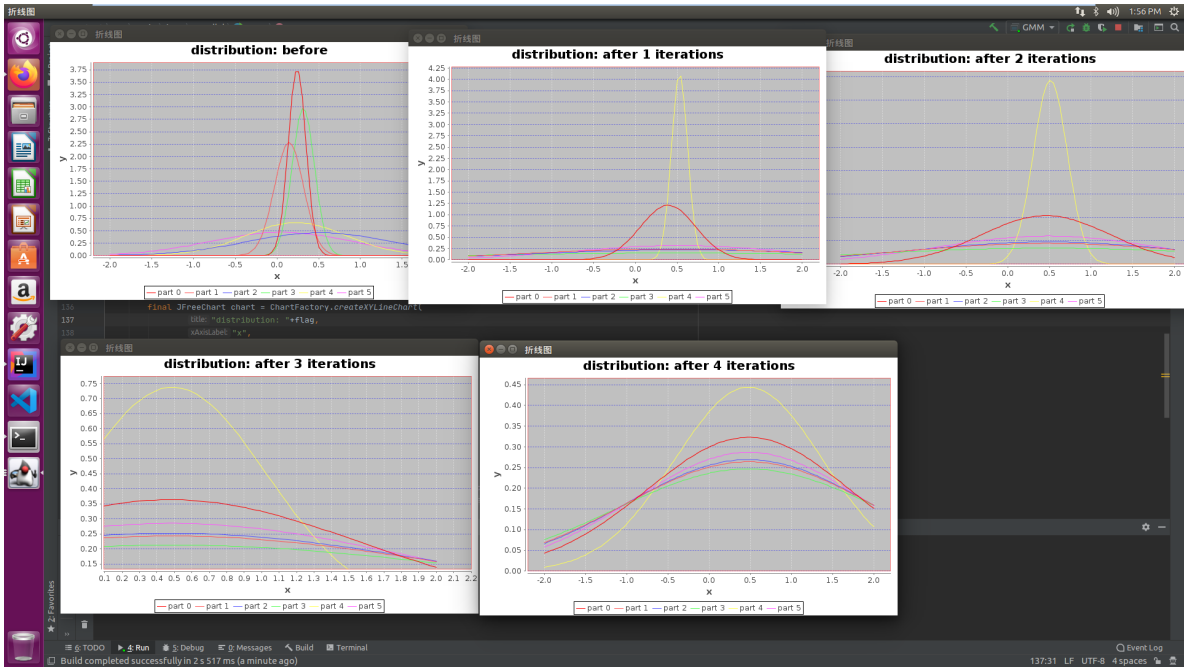
$$N_k = \Sigma_{n=1}^{N} \gamma_{nk}$$

$$\pi_k* = \frac{N_k}{N}$$

$$\mu_k* = \frac{1}{N_k} \Sigma_{n=1}^{N} \gamma_{nk} x_n$$

$$\Sigma_k* = \frac{1}{N_k} \Sigma_{n=1}^{N} \gamma_{nk} (x_n - \mu_k*)(x_n - \mu_k*)^T$$

这里我用一维的随机散点进行仿真实验，首先设置了1000个满足均值为0.5方差为0.3的高斯分布的点，然后用EM算法求它的混合高斯分布。假设这个概率密度分布由六个不同的一维高斯分布组成，可以看到随着四次迭代，分布越来越平滑，mean值逐渐趋近0.5。（这里我是用jfree库来画的图）



EM+MapReduce

可以把N个观测数据$x_1, x_2, \ldots, x_N$,划分成M份数据集，假设这M个子集分别有N_i个观测数据，并且$\sum_{i=1}^{M} N_i = N$。这M份数据将分发给M个mapper，它们将进行如下操作：

[1]：E步中的γ_{nk}计算就应该换成

$$\gamma_{n_i k} = \frac{\pi_k N(X_{n_i}|\mu_k,\Sigma_k)}{\Sigma_{j=1}^{K}\pi_j N(X_{n_i}|\mu_j,\Sigma_j)}$$

[2]：完成$\gamma_{n_i k}$的计算后，为了计算N_k、μ_k*和Σ_k*(参见上面的公式)，我们要事先在Mapper中计算

$S0 = \sum_{n_i=1}^{N_i} \gamma_{n_i k}$，

$S1 = \sum_{n_i=1}^{N_i}(\gamma_{n_i k})x_{n_i}$还有$S2 = \Sigma_{n_i=1}^{N_i}\gamma_{n_i k}(x_{n_i} - \mu_k*)(x_{n_i} - \mu_k*)^T$

[3]：在reducer中，我们就可以计算更新的参数了

$N_k = \sum_{i=1}^{M}\sum_{n_i=1}^{N_i}\gamma_{n_i k} = \sum_{i=1}^{M} S0$

$\pi_k* = \frac{N_k}{N}$

$\mu_k* = \frac{1}{N_k}\sum_{i=1}^{M}\sum_{n_i=1}^{N_i}(\gamma_{n_i k})x_{n_i} = \frac{1}{N_k}\sum_{i=1}^{M} S1$

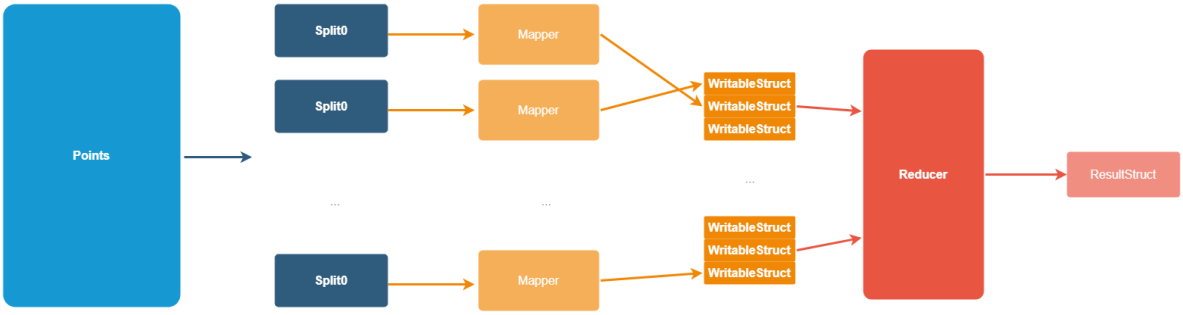
$\Sigma_k* = \frac{1}{N_k}\Sigma_{i=1}^{M}\Sigma_{n_i=1}^{N_i}\gamma_{n_i k}(x_{n_i} - \mu_k*)(x_{n_i} - \mu_k*)^T = \frac{1}{N_k}\Sigma_{i=1}^{M} S2$

实验部分

回顾一下，Mapreduce的工作原理，首先我们要定义Mapper，Reducer（可以在中间设置一个Combiner），尤其是它们的输入输出。

Mapper的输入，毫无疑问，是离散的数据点集，这里我用的是一维的数据点集，所以应该是<LongWritable,Text>的输入（从文本中输入获取点集！点集生成由外部函数完成）。它的输出应该是一个\<IntWritable,WritableStruct>的结构，其中WritableStruct是自定义的Writable子类，应该包含S0,S1,S2这三个变量。

Reducer的输入，就是<IntWritable,WritableStruct>输出的话，应该是一个<IntWritable,ResultStruct>, 而ResultStruct应该包含μ、Σ、π的更新值。



以下是核心代码(MyEM.java)

```
package mole;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;
import java.util.Iterator;

/**
 * 一维EM算法
 * @author hyj
 *1、Mapper for e-m step
 *2、Reducer for m step
 *3、loop
 */
public class MyEM {
    private static int ifLoad = 0;
    private static GMM gmm;

    static {
        try {
            gmm = new GMM(Config.nMix,false);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) throws Exception{
        Configuration conf = new Configuration();
        System.out.println("debug start");
        Job job = Job.getInstance(conf,"MyEM");

        job.setJarByClass(MyEM.class);

        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(WritableStruct.class);
```

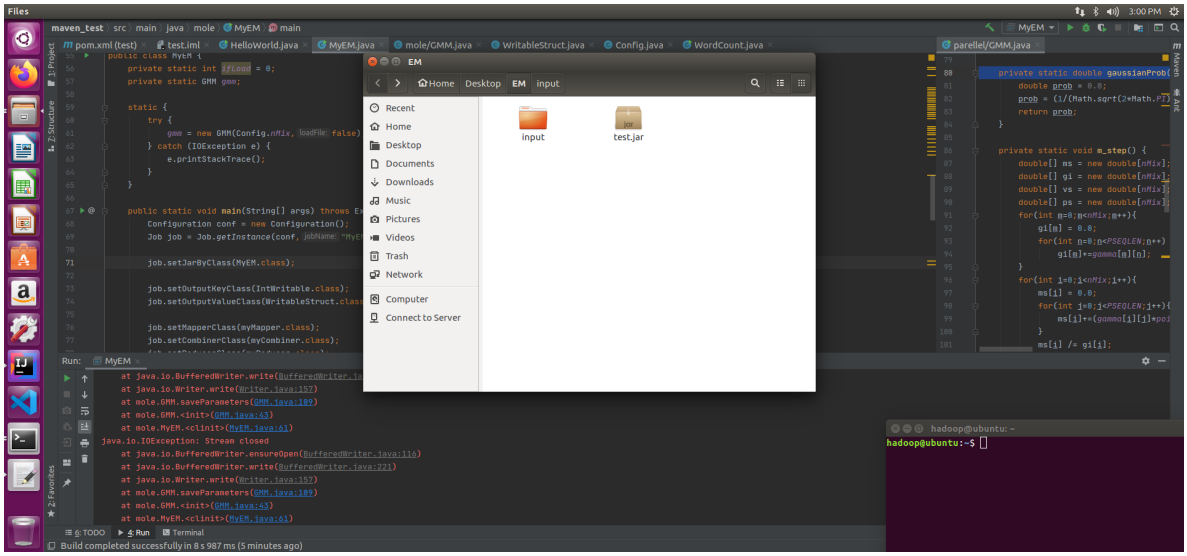
```
        job.setMapperClass(myMapper.class);
        job.setCombinerClass(myCombiner.class);
        job.setReducerClass(myReducer.class);
        job.setNumReduceTasks(1);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job,new Path(args[0]));
        FileOutputFormat.setOutputPath(job,new Path(args[1]));
        ifLoad = Integer.parseInt(args[2]);
        if(ifLoad>0){
            gmm.loadParameters("PARAM.dat");
        }
        job.waitForCompletion(true);
    }
    public static class myMapper extends
Mapper<LongWritable,Text,IntWritable,WritableStruct>{
        private final static IntWritable keyOut = new IntWritable(1);
        @Override
        protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            String token = value.toString();
            double xt = Double.parseDouble(token);
            WritableStruct writableStruct = new WritableStruct();
            double[] gamma = new double[Config.nMix];
            //int PSEQLEN = Config.PSEQLEN;
            int nMix = Config.nMix;
            double[] gmmPosterior = gmm.getPosterior(xt);
            writableStruct.accumulate(gmmPosterior,xt, gmm.getMeans());
            context.write(keyOut,writableStruct);
        }
    }
    public static class myCombiner extends
Reducer<IntWritable,WritableStruct,IntWritable,WritableStruct>{
        @Override
        protected void reduce(IntWritable key, Iterable<WritableStruct> values,
Context context) throws IOException, InterruptedException {
            Iterator<WritableStruct> iter = values.iterator();
            WritableStruct writableStruct = new WritableStruct();
            while(iter.hasNext()){
                WritableStruct curWritableStruct = iter.next();
                writableStruct.accumulate(curWritableStruct);
            }
            context.write(key,writableStruct);
        }
    }
    public static class myReducer extends
Reducer<IntWritable,WritableStruct,IntWritable,Text>{
        @Override
        protected void reduce(IntWritable key, Iterable<WritableStruct> values,
Context context) throws IOException, InterruptedException {
            Iterator<WritableStruct> iter = values.iterator();
            WritableStruct writableStruct = new WritableStruct();
            while (iter.hasNext()){
                WritableStruct curwritableStruct = iter.next();
                writableStruct.accumulate(curwritableStruct);
            }
            gmm.maximize(writableStruct);
```

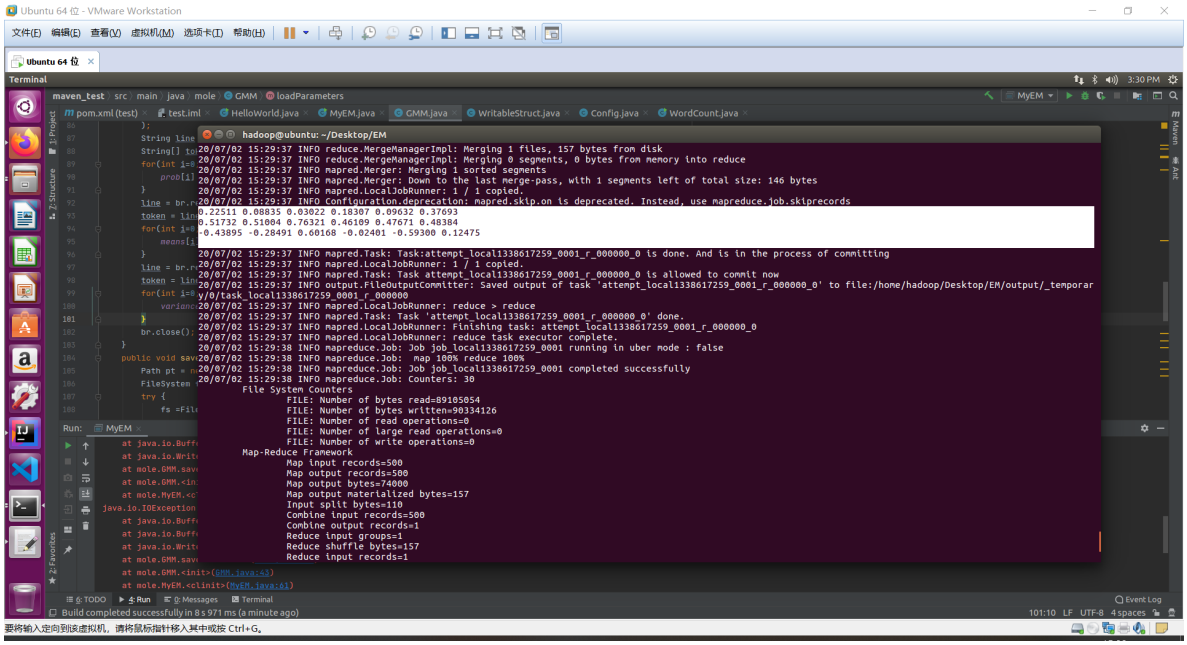
```
                gmm.saveParameters("PARAM.dat");
                System.out.println(gmm.toString());
                Text valueOut = new Text();
                valueOut.set(gmm.toString());
                context.write(key,valueOut);
            }
        }
    }
```

其他代码（GMM.java, WritableStruct.java, Config.java）在压缩包中.

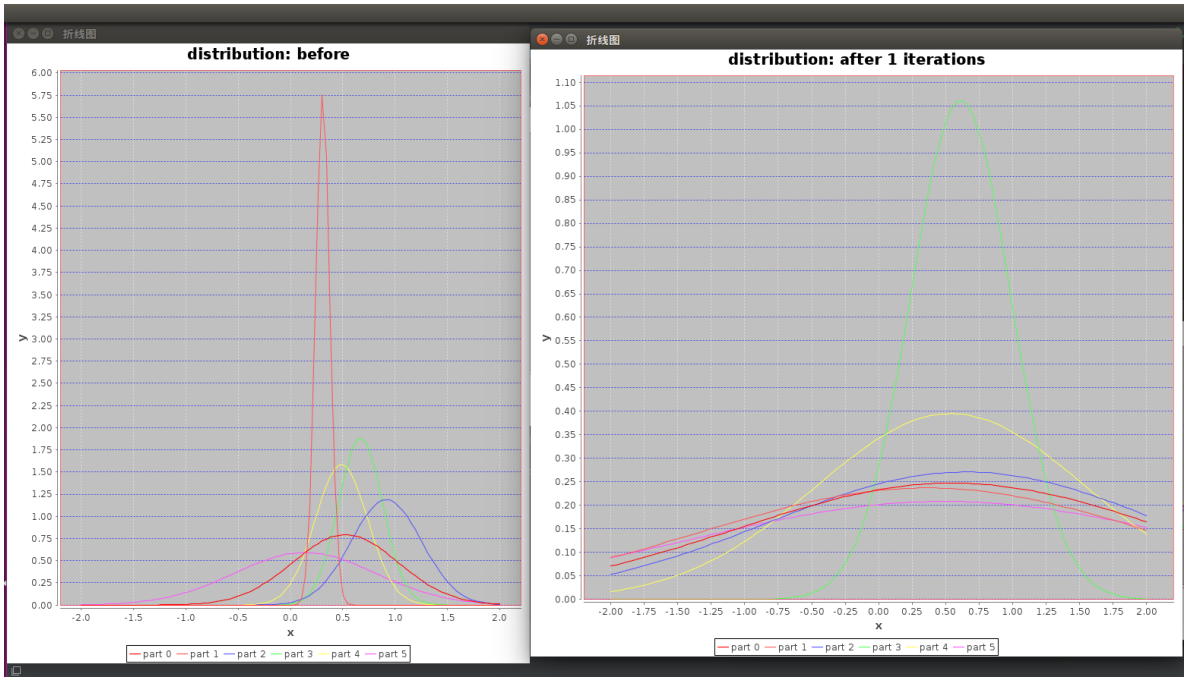
build artifacts生成jar包，我们可以直接上Hadoop试试了



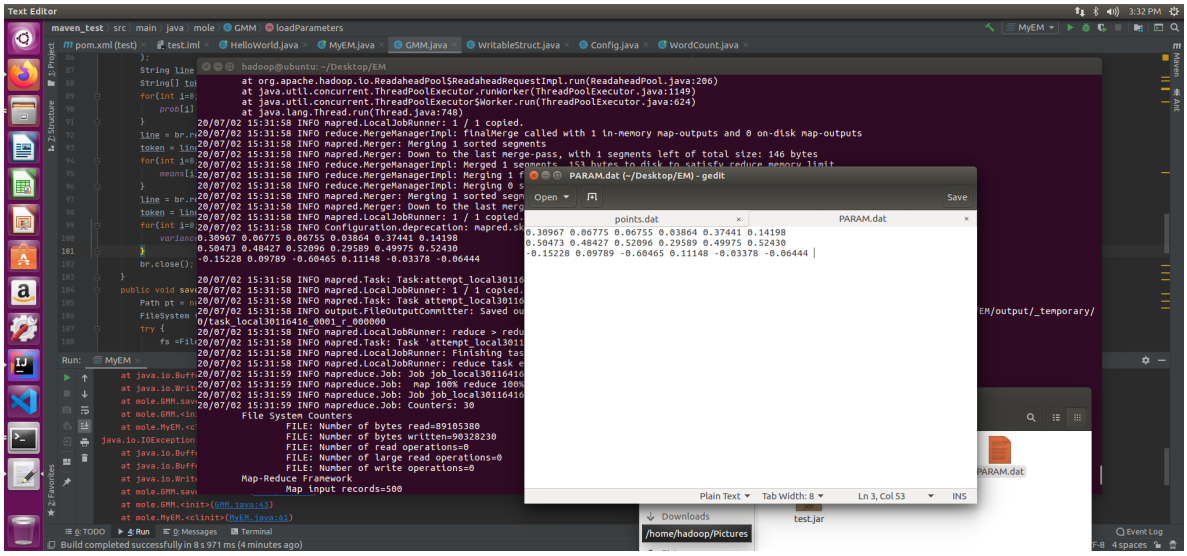
成功!



我们可以看到，几个高斯分布的均值都很靠近0.5，说明这个GMM的效果是好的。这里我还做了个可视化，可以看到EM算法执行前与执行后的对比



这里不妨尝试多次执行EM算法，把`loadFile`改成1后运行数次（每次执行前都要`rm -rf output`）



这里演示的是第五次执行EM算法， 可以从第二行的数据（means）看出，六个不同的高斯分布均值都更接近0.5了，说明这个算法效果是很好的，符合原来数据的特性（生成满足高斯分布、均值为0.5的散点）。

展望

本实验中，为了完成EM算法，我们需要用MapReduce来实现E步和M步的数据处理。而如果我们想进行多个Epoch的训练，则需要多次执行MapReduce过程。实际上，这个步骤可以由Spark来完成，在机器学习的过程里，往往需要多个epoch才能得到令人满意的模型，所以，如果要进一步研究机器学习与云计算的结合，可以往这个方向努力。

参考资料

https://blog.csdn.net/lin_limin/article/details/81048411

https://blog.csdn.net/zhangping1987/article/details/29360785?locationNum=6&fps=1