2017年2月27日 于中国移动集团南方基地

通过协同过滤算法，向行为兴趣相似的用户推送推荐相关服务信息，做到广告和优惠活动的精准推送。

协同过滤是在海量数据中挖掘出小部分与你品味类似的用户，在协同过滤中，这些用户成为邻居，然后根据他们喜欢的东西组成一个排序的目录推荐给你。

协同过滤算法的出现标志着推荐系统的产生，协同过滤算法包括基于用户和基于物品的协同过滤算法。

要实现协同过滤，需要进行如下几个步骤：

1. 收集用户偏好
2. 找到相似的用户或者物品
3. 计算并推荐

从用户的行为和偏好中发现规律，并基于此进行推荐。所以如何收集用户的偏好信息成为系统推荐效果最基础的决定因素。用户有很多种方式向系统提供自己的偏好信息，比如：评分、购买、点击流、页面停留时间等。

减噪：因为用户数据在使用过滤中可能存在大量噪音和误操作，所以需要过滤掉这些噪音。

归一化：不同行为数据的取值相差过能很大，例如用户的查看数据肯定比购买大得多。通过归一化，才能使数据更加准确。

关于相似度的计算很多种方法，比如常用的余弦夹角，欧几里德距离度量，皮尔逊相关系数等等。而如果采用欧几里德度量，那么可以用如下公式来表示相似度：

$$Sim(x,y) = \frac{1}{1+d(x,y)}$$

在计算用户之间的相似度时，是将一个用户对所有物品的偏好作为一个向量，而在计算物品之间的相似度时，是将所有用户对某个物品的偏好作为一个向量。求出相似度后，接下来可以求相似邻居了。

2017年2月28日　于中国移动集团南方基地

现有如下用户购买关系：

	A	B	C	D
1001	√		√	√
1002	√	√		
1003		√		√
1004			√	√

表示用户1001购买过商品A，C和D；用户1002购买过商品A和B

其在文件中的表现形式为Key－Value形式：

```
1001  A
1001  C
1001  D
1002  A
1002  B
⋮
```

将原始文件的值传入MapReduce的map函数，map()函数对原始数据按Key进行分块，同一个Key分到同一个块中。示例数据将被分为4个数据块。然后将map()函数的返回结果传递给reduce()函数，reduce()函数对同一块中的Value进行合并，我们采用','对Value进行分割，即得到如下结果：

```
1001  A,C,D,
1002  A,B,
1003  B,D,
1004  C,D,
```

将以上结果以文本文件的形式保存下来，以备用于下一步的计算。

注：实际生产环境中，原始数据并非我们预想的一样简洁，为了得到我们预期的数据格式，我们常常需要为此再编写解析器和过滤器。

2017年3月1日　　于中国移动集团南方基地

利用上一步的运算结果来计算商品的共现矩阵.

将同一用户购买过的商品一一组合成 key，并且 Value 赋值为 1，提交给 map() 函数进行运算，如下为示例代码：

```
protected void map(LongWritable key, Text value, Context context){
    String[] token = value.toString().split("\t");
    String substring = token[1].substring(0, token[1].length-1);
    String split[] = substring.split(",");
    for(int i=0; i<split.length; i++){
        for(int j=0; j<split.length; j++){
            K.set(split[i] + "\t" + split[j]);
            context.write(K, new IntWritable(1));
        }
    }
}
```

将 map() 的返回结果提交给 reduce() 函数进行运算，reduce() 将相同 Key 的 Value 相加。示例代码如下：

```
protected void reduce(Text key, Iterable<IntWritable> values, Context
    context) throws IOException, InterruptedException{
    int sum=0;
    for(IntWritable v : values){
        sum = sum + v.get();
    }
    context.write(key, new IntWritable(sum));
}
```

我们可以得到如下结果：

A	A	2
A	B	1
A	C	1
A	D	1
B	A	1
B	B	2
B	D	1
C	A	1
C	C	2
C	D	2
D	D	1
D	B	1
D	C	2
D	D	3

2017年3月2日　于中国移动集团南方基地

从上一步的结果我们可以得数学矩阵如下：

	A	B	C	D
A	2	1	1	1
B	1	2	0	1
C	1	0	2	2
D	1	1	2	3

并且我们可以发现共现矩阵一定是对称矩阵。但此时的数据并不能直接参与下一步的运算，我们需要将矩阵进行转化为Key-Value的形式，此时我们还需要创建一个实体类ProductVector用于存放商品和共现次数。

map()函数如下：

```
protected void map(Text key, Text value, Context context){
    String[] split = value.toString().split("\t");
    v.setProductId(split[0].trim());
    v.setQuantity(Double.parseDouble(split[1].trim()));
    context.write(key, v);
}
```

reduce()函数如下：

```
protected void reduce(Text key, Iterable<ProductVector> values, Context context){
    Set<String> l = new TreeSet<String>();
    l.clear();
    for(ProductVector v : values){
        l.add((new ProductVector(v)).toString());
    }
    context.write(key, new Text(l.toString()));
}
```

我们最终可以得到如下结果：

A　[A:2.0, B:1.0, C:1.0, D:1.0]

B　[A:1.0, B:2.0, D:1.0]

C　[A:1.0, C:2.0, D:2.0]

D　[A:1.0, B:1.0, C:2.0, D:3.0]

2017年3月3日　于中国移动集团南方基地

至此，我们已经得到了商品间共现矩阵，还需要每件商品的用户参考矩阵。参考矩阵需要从原始数据中计算得出。其中，我们还需定义一个存放参考关系的实体类 Preference。

map() 函数的代码示例如下：

```
protected void map(LongWritable key, Text value, Context context){
        Text k = new Text();
        Preference v = new Preference();
        Parser parser = new Parser();
        parser.parse(value.toString());
        v.setId(parser.getClient());
        v.setPref(new DoubleWritable(parser.getPreference().get()));
        k.set(parser.getProduct());
        context.write(k, v);
}
```

reduce() 函数的代码示例如下：

```
protected void reduce(Text key, Iterable<Preference> values, Context context){
        Set<String> l = new TreeSet<String>();
        Text value = new Text();
        l.clear();
        for(Preference v: values){
            l.add((new Preference(v)).toString());
        }
        value.set(l.toString());
        context.write(key, value);
}
```

可得到如下结果：

A　[1001:1.0,　1002:1.0]

B　[1002:1.0,　1003:1.0]

C　[1001:1.0,　1004:1.0]

D　[1001:1.0,　1003:1.0,　1004:1.0]

2017年3月6日　于中国移动集团南方基地

我们可以将参考矩阵转化成数学矩阵形式：

	1001	1002	1003	1004
A	1	1	0	0
B	0	1	1	0
C	1	0	0	1
D	1	0	1	1

在矩阵相乘之前，我们还需要做一些准备，编写一个解析器和一个自定义类型。

自定义类型 VectorWritable 用于存放和获得 Preference 类型对象，其需要实现接口 Writable 和 Iterable<Preference>，需要重写 toString()，write(DataOutput out)，readFields(DataInput in) 和 Iterator<Preference> 方法和一些自定义方法如 add()，remove()，clear() 等。其还有一个重要的成员变量是 Set<Preference> s，用于存放 Preference 对象。

解析器 VectorParser 用于解析 value 并将其封装成 Preference 对象，存放在 VectorWritable 对象中。示例代码如下：

```
public class VectorParser {
    private VectorWritable vector = new VectorWritable();
    private boolean isValid = false;
    public void parse(String value){
        String substring = value.substring(1, value.length()-1);
        String[] split = substring.split(",");
        if(split.length > 0){
            for(int i=0; i<split.length; i++){
                String[] element = split[i].split(":");
                Preference p = new Preference();
                p.setId(element[0].trim());
                p.setPref(Double.parseDouble(element[1].trim()));
                vector.add(p);
            }
            isValid = true;
        }
    }
    ⋮
}
```

2017年3月7日　于中国移动集团南方基地

到目前为止，我们的准备工作已经完成，现在即将处理和解析数据。首先，对参考矩阵和共现矩阵分别进行map()函数运算。

map()函数示例代码如下：

```
protected void map(LongWritable key, Text value, Context context){
        String[] split = value.toString().split("\t");
        TextTuple k = new TextTuple();
        VectorWritable v = new VectorWritable();
        protected void map
        VectorParser parser = new VectorParser();
        v.clear();
        if(split != null && split.length == 2){
            k.setText(split[0]);
            k.setTag("1");
            parser.parse(split[1]);
            if(parser.isValid()){
                v = parser.getVector();
            }
            context.write(k, v);
        }
}
```

以上代码为共现矩阵的map()函数，只要将key的tag属性设置为0就可以成为参考矩阵的map()函数。

现在，我们需要设置分区和分组的类，使分区更加均匀，不会出现数据倾斜的情况，使reduce()函数均衡负载。还需要对组内数据进行排序。

分区函数的示例代码如下：

```
public int getPartition(TextTuple key, Text value, int numPartitions){
    return Math.abs(key.getText().hashcode() * 127) % numPartitions;
}
```

比较器示例代码如下：

```
public int compare(WritableCompare a, WritableCompare b){
    TextTuple o1 = (TextTuple) a;
    TextTuple o2 = (TextTuple) b;
    return o1.getText().compareTo(o2.getText());
}
```

2017年3月8日　　于中国移动集团南方基地

现在我们就可以在 reduce() 函数中实现矩阵相乘了。

reduce() 函数示例代码如下：

```
protected void reduce (TextTuple key, Iterable<VectorWritable> values,
        Context context) throws IOException, InterruptedException {
    Text k = new Text();
    Preference v = new Preference();
    List<Preferenee> client;
    List<Preference> product;
    protected void re
    Iterator<VectorWritable> iterator = values.iterator();
    client = iterator.next().toList();
    product = iterator.next().toList();
    for (Preference c : client){
        k.set (c.getId);
        for (Preference p : product){
            v.setId (p.setId());
            v.setPref (p.getPref().get() * c.getPref().get());
            context.write(k, v);
        }
    }
}
```

其输出结果为：

1001　[A:4.0,　B:2.0,　C:5.0,　D:6.0]

1002　[A:3.0,　B:3.0,　C:1.0,　D:2.0]

1003　[A:2.0,　B:3.0,　C:2.0,　D:4.0]

1004　[A:2.0,　B:1.0,　C:4.0,　D:5.0]

2017年3月9日　于中国移动集团南方基地

用数学方法模拟以上过程，即

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 1 & 2 & 3 \end{bmatrix} \end{array} \times \begin{array}{c} \begin{array}{cccc} 1001 & 1002 & 1003 & 1004 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ A \\ B \\ C \\ D \end{array}$$

$$\Downarrow$$

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} 1001 & 1002 & 1003 & 1004 \end{array} \\ \begin{bmatrix} 4 & 3 & 2 & 2 \\ 2 & 3 & 3 & 1 \\ 5 & 1 & 2 & 4 \\ 6 & 2 & 4 & 5 \end{bmatrix} \end{array}$$

现在我们要做的就是将用户购买过的商品剔除。

map()函数的示例代码如下：

```
protected void map(LongWritable key, Text value, @Context context){
    TextTuple k = new TextTuple();
    Text v = new Text();
    String[] split = value.toString.split(" ");
    k.setText(split[0] + "\t" + split[1]);
    k.setTag("1");
    v.set(split[2]);
    context.write(k, v);
}
```

reduce函数的示例代码如下：

```
protected void reduce(TextTuple key, Iterable<Text> values, Context context){
    int count = 0;
    for(Text v: values){
        count++;
        l.add(v);
    }
    if(count==1){
        context.write(key.getText(), l.get(0));
    }
}
```

2017年3月10日　于中国移动集团南方基地

经过数据剔除，我们可以得到如下结果：

1001　[B:2.0]
1002　[C:1.0,　D:2.0]
1003　[A:2.0,　C:2.0]
1004　[A:2.0,　B:1.0]

现在我们可以从结果中找出推荐数值最大的商品与用户一一对应，并将其保存到数据库中。

map函数示例如下：

```
protected void map(LongWritable key, Text value, Context context){
    KeyTuple k = new KeyTuple();
    ValueTuple v = new ValueTuple();
    String[] split = value.toString().split("\t");
    k.setText((split[0]));
    k.setTag(split[2]);
    v.setProductId(split[1]);
    v.setPref(Double.parseDouble(split[2]));
    context.write(k, v);
}
```

reduce函数示例如下：

```
protected void reduce(KeyTuple key, Iterable<ValueTuple> values, Context context){
    RecommendationDB k = new RecommendationDB();
    int count = 0;
    for(ValueTuple value : values){
        if(count < 1){
            k.setClient(key.getText());
            k.setProduct(value.getProductId());
            k.setPreference(value.getPref());
            context.write(k, NullWritable.get());
            count++;
        } else
            break;
    }
}
```