在 WKStream 原有算法的基础上加入香农熵的实验报告

在原本的算法中，算法的提出者给数值型属性按照某种规则赋予了不同的权重，而对于不同的非数值型属性认为是权重相同的，因此在原有算法的基础上，我提出了一种基于香农熵给非数值型属性赋予一定的权重，希望以此增加数据的可分性，使聚类效果更加明显。

代码中需要修改的代码以及代码分析：

类似于计算原来的数值型属性的 Nonuniform，首先我在 Formula 这个 class 中定义计算香农熵的公式

```
public static double Entropy(int[] arr,int n) {
    double result=0;
    Vector v=new Vector();
    for(int i=0;i<n;i++) {
        if(!v.contains(arr[i])) {
            v.add(arr[i]);
        }
    }
    for(int i=0;i<v.size();i++) {
        int cnt=0;
        int cur=Integer.parseInt(v.get(i).toString());
        for(int j=0;j<n;j++)
        {
            if(cur==arr[j]){
                cnt++;
            }
        }
        double p=cnt*1.0/n;
        result-=p*Math.log(p)/Math.log(2);
    }
    return result;
}
```

通过使用小数据集验证，证明了这个函数的代码是正确的，代码运行的结果与理论计算的结果一样。

```
public static double D2(DataPoint x, DataPoint y) {
    int Ist = 0;// Intersection
    int Uni = ConstantVar.NCat;// Union
    double res = 0;
    for (int i = 0; i < ConstantVar.NCat; ++i)
        if (x.ca[i] == y.ca[i])
            ++Ist;
    res = (double) Ist / Uni;
    return 1 - res;
}
```

```
public static double D2(DataPoint x, DataPoint y,double[] h) {
    double Ist = 0;// Intersection
    double Uni = 0;// Union
    double res = 0;
    for (int i = 0; i < ConstantVar.NCat; ++i) {
        if (x.ca[i] == y.ca[i])
            Ist+=h[i];
        Uni+=h[i];
    }
    res = (double) Ist / Uni;
    return 1 - res;
}
```

（左图为原算法，有图为增加香农熵以后的算法）

然后是重写计算数据样本的非数值型属性的距离，在上图中，注意到在原来的算法中不同属性的权重都是 1，而在新算法中，通过根据属性取值计算出来的香农熵 h 给不同的属性赋予不同的权重。

除此以外还需要进行算法一开始对熵值的初始化等等。

参数分析：

1. $$D(\mathbf{x},\mathbf{y}) = (1-\lambda)D_1(\mathbf{x},\mathbf{y}) + \lambda D_2(\mathbf{x},\mathbf{y})$$

综合计算数值型属性与非数值型属性的距离时的平衡系数λ，注意到，在原来的算法中，λ的取值为 0.5，而非数值型属性的权值都是 1；在新算法中，权值是根据该属性的香农熵/总的香农熵来确定的，因此是一个小于 1 的数，对于某些属性，权重的取值比 1 还要小得多。因此，假设在原来的算法中λ=0.5 这一取值是较为合适的，在新算法中这一取值就不合适了，因为在数值型属性距离不变的前提下，非数值型属性的距离整体变小了，这就导致了两种距离的不均衡，因此λ应该取更大的值，我尝试过 0.6，0.7，0.8，结果在后面分析。

2. 还有一些对实验结果产生影响的参数：各种阈值、簇的个数等等。

结果分析：

在做这个改进实验的过程中，梁文斌师兄也有跟我讨论以及一起分析实验结果，因此我后来才知道，这份代码当时是用来处理一个项目的，当时好像是一个多分类的问题，而在老师给的测试数据中，这是一个二分类问题，因此代码有一些常数是需要修改的，比如说最终要聚成多少类。当时我问了师兄，师兄对于代码的一些细节已经想不清了，所以我只能根据自己的理解对代码原来的一些常数进行修改，修改了以后，出来的结果确实是两类，然后我使用 NMI 算法对聚类结果与实际聚类进行评测，发现结果非常差，甚至还达不到 0.01。后来我按照代码流程重新理了一遍思路，觉得没有错。最终我尝试了将原来的算法通过修改参数变成二分类的聚类，发现结果也是很差，只有 0.01 几，这个时候我在想，可能已经不是算法的处理过程的问题，而是一些参数的问题。我还注意到的一个现象就是，对于原来的聚类成 5 类，代码的运行速度很快，而对于后来的二分类，代码的运行速度就很慢，我觉得是一些阈值处理的问题,而在算法过程中,涉及的阈值也不少。由于我没有经历原来的实验过程，对于这些阈值的确定标准不清楚,在不清楚算法细节的时候贸贸然进行修改也不可能有什么很大的改进，而且梁文斌师兄对原来的很多算法细节都已经不记得了，所以我就没有继续做下去了。

最终的实验结果是，在原来的阈值的基础上，通过修改距离的平衡权重 λ，最终达到的结果与原来算法差不了多少，没有明显的提高，甚至有时计算出来的 nmi 值还会更低。而由于阈值以及一些参数的不合适，最终得出的整体效果非常差，计算出来的 NMI 大概只有 1%左右。