中山大学数据科学与计算机学院

移动信息工程专业-数据仓库与数据挖掘

本科生实验报告

教学班级	1515	专业（方向）	软件工程
学号	15352334	姓名	吴佳卫

一、 实验题目

图聚类

二、 实验内容

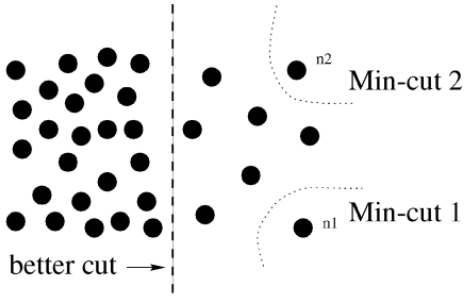
1. 算法原理

正则化割 Normalized cut

讨论到正则化割算法就要先讨论另一个算法：最小割算法。而正则化割算法正是最小割算法的优化。在最小割公式中：

$$\mathrm{cut}(\mathrm{A},\mathrm{B}) = \sum_{u\in A, v\in B} w(u,v)$$

通过最小割公式构建两点之间的相似度可以很好地划分出权重更大的点。但是最小割算法的问题在于算法有可能会划分出单个的离群点，显然这样是无法进行成功的聚类的。



因此正则化割算法提出了相似度函数的优化：

$$\mathrm{assoc}(\mathrm{A},\mathrm{v}) = \sum_{u\in A, t\in V} w(u,t)$$

$$\mathrm{Ncut}(\mathrm{A},\mathrm{B}) = \frac{cut(A,B)}{assoc(A,v)} + \frac{cut(A,B)}{assoc(B,v)}$$

其中$\mathrm{assoc}(\mathrm{A},\mathrm{v})$ 计算的是 A 中所有点与图中所有点相连的权重。可以看到如果想保证$\mathrm{Ncut}(\mathrm{A},\mathrm{B})$的值比较大，我们需要确保$cut(A,B)$的值比较大，而$assoc(A,v)$与$assoc(B,v)$相对较小。前者保证划分出的 A,B 两个簇有较高的权重，而后者保证 A 与 B 两个簇不会太小。

随后我们基于 Normalized cut 的公式进行 k-路划分，也就是与上周实验谱聚类的第一步相

同，构造标准化拉普拉斯矩阵，解出前 K 小特征向量。然后使用 K-means 进行聚类聚类划分，产生 k'个新的类

。

在这里 k'的取值应该是一个在 100 以内尽量大的数。我们可以这样理解：k'的值越大，我们从特征矩阵中提取到的信息就会更多，这是更加有利于我们进行后面的聚类的。

最后一步是根据这 k'个类聚合成 k 个类，聚合的法则如下：每次选取两个使 Ncut 最小的两个类进行合并。

Louvain

Louvain 算法是基于层次聚类与模块度 modularity 的一种图聚类算法。层次聚类的基本思想是一种自下而上的划分。也就是说层次聚类初始化所有的点为不同的簇，然后每次合并两个最相近的簇，直到完成聚类。而在这里我们衡量两个簇之间的距离使用的指标是模块度增益：

$$\Delta Q = \left[\frac{\sum in}{m} + \left(\frac{k_{i,in}}{2m}\right)^2\right] - \left[\frac{\sum in}{2m} - \left(\frac{\sum tot}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

几个变量的定义如下：

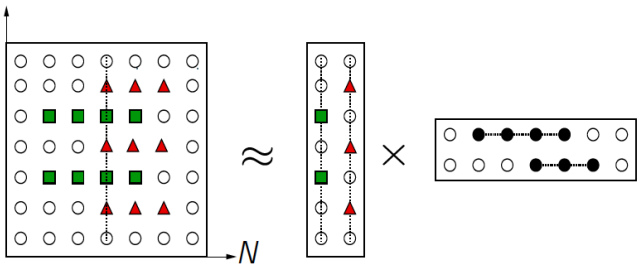
$\sum in$	社区 C 内部全部边的权重之和	每一条边只需要计算一次
$\sum tot$	社区 C 内全部节点的边权重之和	需要注意的是社区内的每条边都被计算了两次
k_i	与节点 i 相连的所有边的权重之和	只要计算点 i 的度数即可，一种更加简单的计算方法为 $k_{i,in} = \sum tot - 2\sum in$
$k_{i,in}$	节点 i 和社区 C 内部节点的权重之和	注意要乘 2
m	整个网络中边的权重之和	总边数，是一个定值

模块度本身可以被用来衡量一个社区的划分是不是相对比较好的结果。一个相对好的结果在社区内部的节点相似度较高，而在社区外部节点的相似度较低。基于这一个特征，每次我们遍历每一个节点以及其所有邻居，并且尝试将节点划分入可以使整体模块度增益最大的邻居所属的簇中。

然后我们将一个簇融合为一个新的节点（内部的边看做自环，节点之间的边权重为原来两个簇之间的边权重之和）

非负矩阵分解 NMF

非负矩阵分解的核心思想是将原邻接矩阵分解为两个矩阵 U 和 V。其中$UV^T \to A$。在这里可以把 U 矩阵理解为特征矩阵，V 矩阵为权值矩阵。



而我们有一个目标函数可以衡量划分的结果，也就是：

$$L_{LSE}(A, UV^T) = \|A - UV^T\|_F^2$$，其中：

$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$也就是弗罗贝尼乌斯范数 Frobenius norm

根据这一个目标函数，有学者提出了 U 和 V 矩阵的更新公式：

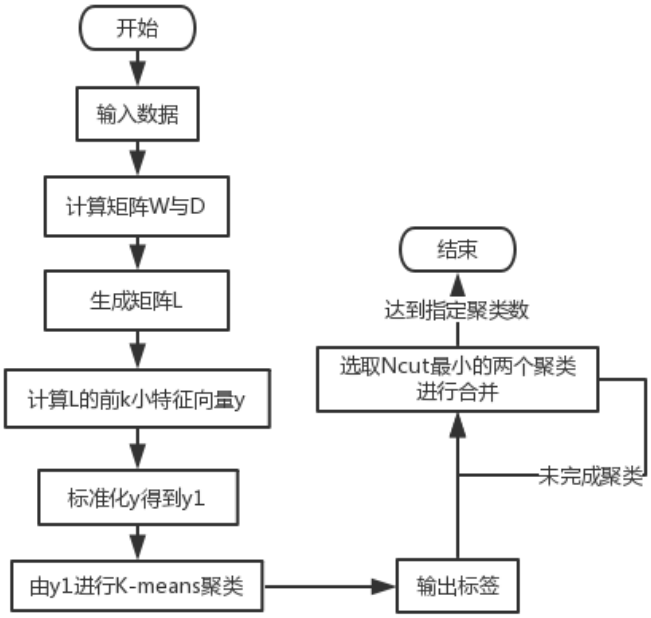
$$(U)_{ij} \leftarrow (U)_{ij}\frac{(AV)_{ij}}{(UV^TV)_{ij}}$$

$$(V)_{ij} \leftarrow (V)_{ij}\frac{(A^TU)_{ij}}{(VU^TU)_{ij}}$$

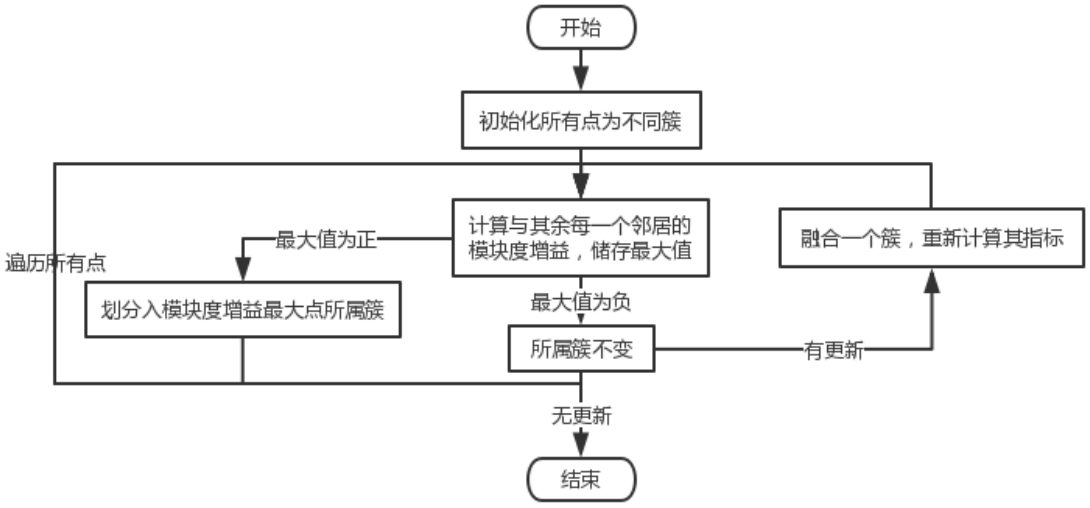
只要使用以上的更新公式迭代更新 U 和 V 矩阵就可以了。最后我们取 V 矩阵第 i 行的最大值V_{ij}，也就是第 i 个数据属于类 j。这一步判断基于的原理是 V 矩阵是一个权值矩阵，因此最大的权值可以表示这一个点 i 对于簇 j 的特征是最明显的，因此可以完成划分。

2. 算法流程

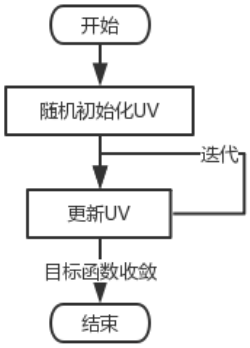
正则化割 Normalized cut



Louvain



非负矩阵分解 NMF



3. 关键代码截图

正则化割 Normalized cut

算法的第一步与上周的谱聚类相同，通过 W 和 D 矩阵计算标准化拉普拉斯矩阵，然后计算前 K 小特征向量。然后接入一个 K-means 聚类算法，聚为 k'个类，由于篇幅问题 K-means 算法就不贴代码了。

```
n = size(adj,1);
W = adj;
%W = W - diag(diag(W));
D =diag(sum(W'));

L = D^(-.5)*W*D^(-.5);

[eigVectors,eigValues] = eig(L);
[~, ind] = sort(diag(eigValues), 'ascend');
nEigVec = eigVectors(:,ind(1:K));

U=zeros(size(nEigVec,1),K);
for i=1:size(nEigVec,1)
    tt = sqrt(sum(nEigVec(i,:).^2));
    if tt~=0
        U(i,:) = nEigVec(i,:) ./ tt;
    end
end
```

最后我们得到的是将所有点聚为 k'个点的类标

```
gg = repmat(mat(j,:), K, 1);
gg = gg - clusters;    % norm:计算向量模长
for n=1:K
    tt(:,n) = norm(gg(n,:));
end
[~, minIdx] = min(tt);
c(j) = minIdx;
```

也就是 c。

然后我们两两计算出比较聚类 i 与聚类 j 的 Ncut。

```
cutAB = zeros(sample_number,1);
assocAV = zeros(sample_number,1);
for i=1:sample_number
    for j=1:sample_number
        if adj(i,j) == 1
            assocAV(i) = assocAV(i) + 1;
        end
        if adj(i,j) == 1 && i~=j && c(i) ~= c(j)
            cutAB(i) = cutAB(i) + 1;

        end
    end
end
for i = 1:sample_number
    if assocAV(i) ~= 0
        Ncut(q,p) = Ncut(q,p) + cutAB(i)/assocAV(i);
    end
end
clear c;
```

取每次最小的 Ncut，然后合并这两个聚类，直到最后只剩下 k 个聚类。

```
[a,min_row] = min(Ncut);
[~,min_col] = min(a);

for i = 1:sample_number
    if c_temp(i) == min_col
        c_temp(i) = min_row(min_col);
    end
end

K = K - 1;
clear Ncut;
```

Louvain

考虑到 Louvain 算法对数据结构的依赖，这里使用 c++实现。

首先定义了一个簇的结构，包括内部的元素 ele，聚类本身的标签 label，以及三个指标$\sum in$，$\sum tot$ 与k_i。因为考虑到这三个值在每一次迭代内都是相对固定的，因此把他们存下来可以节省时间的开销。

```
struct cluster
{
    vector<int> ele;
    int label;
    double in;
    double tot;
    double ki;
};
```

计算节点 i 和社区 c 的模块度增益的函数如下，由于三个值已经在簇中保存，因此在这里只需要计算$k_{i,in}$就可以了（注意要乘 2 的）

```
double cal_delta_Q(int i, int c)
{
    double kiin = 0;
    for(int j = 0 ; j < clu[c].ele.size(); j ++)
    {
        kiin += adj[i][clu[c].ele[j]];
    }
    kiin *= 2;
    double delta_Q = 0;
    double tmp_0 = (clu[c].in + kiin) / degree_sum;
    double tmp_1 = pow(((clu[c].tot + clu[c].ki) / (2 * degree_sum)), 2);
    double tmp_2 = clu[c].in / (2 * degree_sum);
    double tmp_3 = pow((clu[c].tot / (2 * degree_sum)), 2);
    double tmp_4 = pow((clu[c].ki / (2 * degree_sum)), 2);
    delta_Q = (tmp_0 - tmp_1) - (tmp_2 - tmp_3 - tmp_4);
    return delta_Q;
}
```

遍历所有点并计算与邻居的模块度增益，并且完成分配。

```
bool merge0()
{
    bool update = false;
    for(int i = 0; i < num; i ++)
    {
        double max_delta_Q = -1;
        double max_point;
        for(int j = 0 ; j < num; j ++)
        {
            if(i != j && connected(i, j))
            {
                double tmp_delta_Q = cal_delta_Q(i, j);
                if(tmp_delta_Q > max_delta_Q)
                {
                    max_delta_Q = tmp_delta_Q;
                    max_point = j;
                }
            }
        }
        if(max_delta_Q != -1)
        {
            update = true;
            for(int j = 0; j < clu[i].ele.size(); j ++)
            {
                Label[clu[i].ele[j]] = Label[clu[max_point].ele[0]];
                //cout<<"merge"<<i<<" "<<max_point<<endl;
            }

        }
    }
    return update;
}
```

在完成上面的操作后，重新生成簇。

```
clu.clear();
vector<int> new_label;
for(int i = 0; i < Label.size(); i ++)
{
    int j = 0;
    while(j < new_label.size())
    {
        if(new_label[j] == Label[i]) break;
        j ++;
    }
    if(j == new_label.size())
    {
        new_label.push_back(Label[i]);
        cluster c;
        clu.push_back(c);
        clu.back().ele.push_back(i);
        clu.back().label = clu.size() - 1;
    }
    else
    {
        clu[j].ele.push_back(i);
    }
}
num = new_label.size();
```

重新计算三个指标$\sum in$，$\sum tot$ 与k_i。

```
for(int i = 0; i < clu.size(); i ++)
{
    clu[i].in = 0;
    for(int j = 0 ; j < clu[i].ele.size(); j ++)
    {
        for(int k = j + 1 ; k < clu[i].ele.size(); k ++)
        {
            clu[i].in += adj[clu[i].ele[j]][clu[i].ele[k]];
        }
    }
    clu[i].tot = 0;
    for(int j = 0; j < clu[i].ele.size(); j ++)
    {
        clu[i].tot += degree[clu[i].ele[j]];
    }
    clu[i].ki = clu[i].tot - 2 * clu[i].in;
    for(int j = 0; j < clu[i].ele.size(); j ++)
    {
        Label[clu[i].ele[j]] = i;
    }
}
```

非负矩阵分解 NMF

NMF 的代码很简单，首先是一个目标函数

```
function lse = LSE(M)

lse = 0;
[n,~] = size(M);
for i=1:n
    for j=1:n
        lse = lse + M(i,j);
    end
end

end
```

然后在目标函数收敛前不断更新 U 矩阵和 V 矩阵

```
function [U,V] = nmf(A,k)

[n,~] = size(A);
U = abs(rand(n,k));
V = abs(rand(n,k));
lse0 = 0;
lse1 = LSE(A-U*V');
iter = 0;
while(abs(lse0 - lse1) > 0.0000000001)
    n_U = U .* (A*V) ./ ((U*V')*V);
    n_V = V .* (A'*U) ./ ((V*U')*U);
    U = n_U;
    V = n_V;
    lse0 = lse1;
    lse1 = LSE(A-U*V');
    disp(lse1);
    iter = iter +1;
end

[~,index]=max(V,[],2);
dlmwrite('E:\Jason\学术\Data Mining\Lab\Lab2\code\result.txt',index);
disp(iter);

end
```

4. 创新点&优化

提供了一种使用特征矩阵的思路（标准化割）也就是在数据预处理的时候，将特征矩阵 fea 与邻接矩阵 adj 进行融合，融合的方法与上周谱聚类生成邻接矩阵的思路相同，也就是：

$$W_{ij} = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

```
function [adj_weight] = cal_adj(adj,fea)

sigma = 1;
n = size(adj,1);
W = zeros(n,n);
for i = 1:n
    for j = n:-1:i
        if adj(i,j) == 1
            W(i,j) = exp(-norm(fea(i,:)-fea(j,:))^2/2/sigma);
        end
        W(j,i) = W(i,j);
    end
end

adj_weight = W;
```

这个生成的结果直接替换掉原来的邻接矩阵就可以了。

```
function [C] = normalized_cut(adj,target_K)

adj = cal_adj(adj,fea);

K = 10;
c = kdivide(adj,K);
c_temp = c;
sample_number = size(c,1);
```

三、 实验结果及分析

以下的结果直接在运行代码后，导入 clustermeasure 函数进行四个指标的分析，结果整理如下：

正则化割 Normalized cut

	ACC	NMI	PUR
cornell	0.4103	0.0634	0.6205
texas	0.4813	0.1153	0.5882
washington	0.4739	0.0629	0.8261
wisconsin	0.4189	0.0439	0.7321

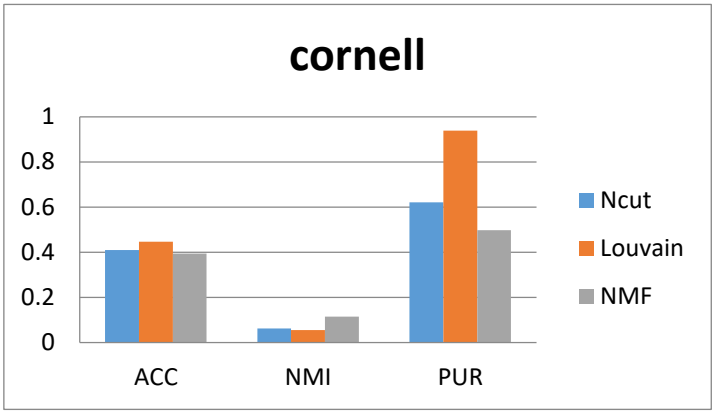
Louvain

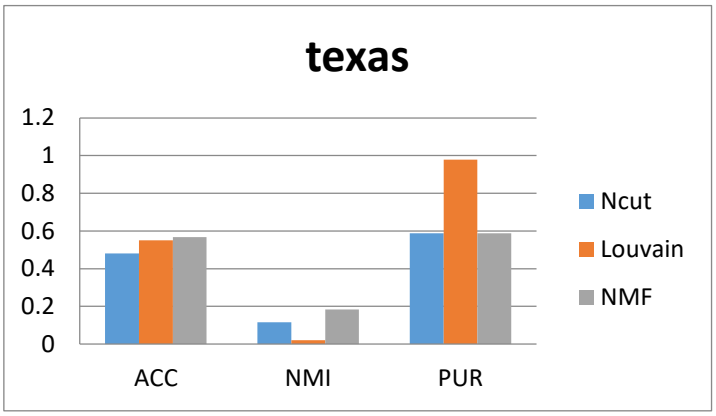
	ACC	NMI	PUR
cornell	0.4462	0.0553	0.9385
texas	0.5508	0.0211	0.9786
washington	0.4609	0.0651	0.9348
wisconsin	0.4604	0.0337	0.9472

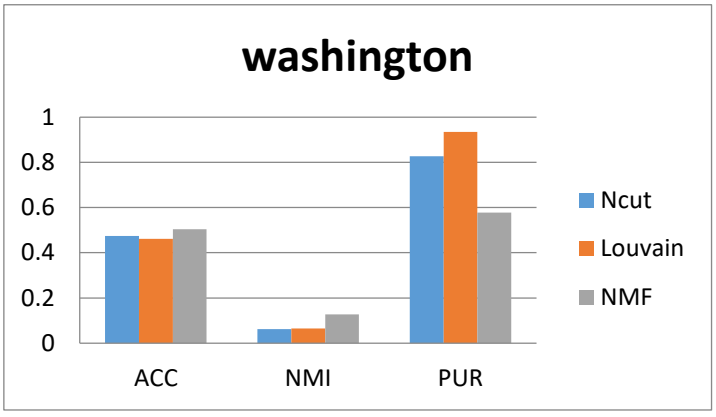
非负矩阵分解 NMF

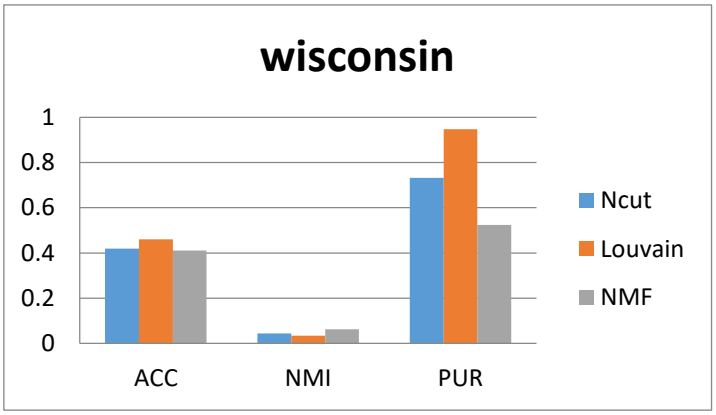
	ACC	NMI	PUR
cornell	0.3949	0.1150	0.4974
texas	0.5668	0.1830	0.5882
washington	0.5043	0.1273	0.5783
wisconsin	0.4113	0.0628	0.5245

然后我们进行算法之间的横向比较







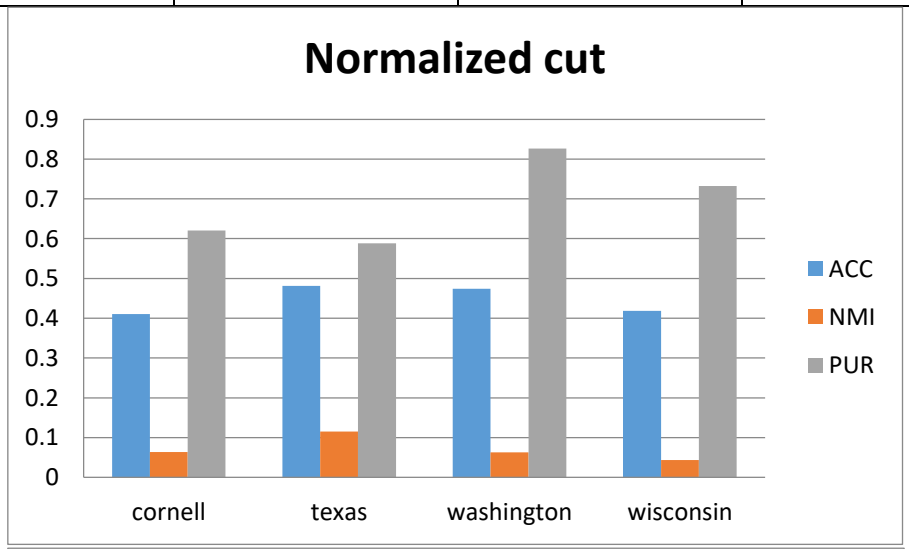


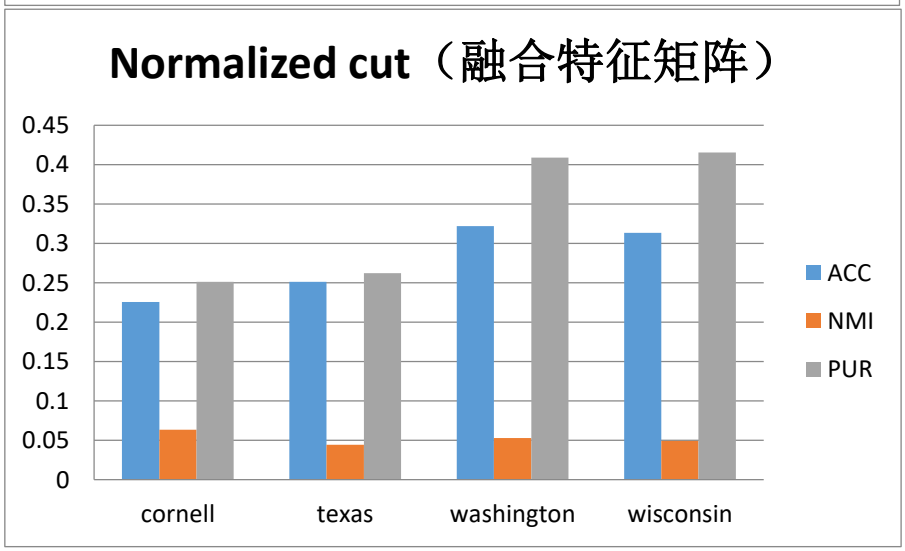
可以看出本次试验三个算法的 ACC 都在 0.5 上下，而 NMI 都在 0.1 左右表现较差，PUR 参差不齐。总体俩说 Louvain 的优势是更加显著的。

最后进行融合了特征矩阵的 Ncut 算法分析

正则化割 Normalized cut（融合特征矩阵）

	ACC	NMI	PUR
cornell	0.2256	0.0633	0.2513
texas	0.2513	0.0443	0.2620
washington	0.3217	0.0529	0.4087
wisconsin	0.3132	0.0490	0.4151





可以观察到结果并没有变得更好，推测是因为特征矩阵是一个 01 矩阵，因此在我们融合特征矩阵和邻接矩阵的时候对原本的邻接矩阵差生了很大的干扰，因此使最后的效果也受到了影响。