# 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

Network Embedding

## 二、 实验内容

### 1. 算法原理

### M_NMF

M_NMF 算法与上次实验的 NMF 算法思路比较相近，也就是通过矩阵分解的方法从原来的一些表示网络的矩阵中分解出特征矩阵。而 M_NMF 算法的优点在于他可以很好的挖掘出网络中更加深层次的信息。

为了构建相似矩阵 S，M_NMF 算法使用了一阶相似度$S^{(1)}$（邻接矩阵）和二阶相似度$S^{(2)}$（余弦相似度）$S^{(2)}=\frac{N_iN_j}{\|N_i\|\|N_i\|}$，其中$N_i$是$S^{(1)}$的第 i 行。

同时，我们使用模块度的概念来刻画社区的信息，模块度 Q 可以记为$\mathrm{tr}(H^T\mathrm{BH})$，其中 H 是社区指示矩阵。最后，我们的目标函数可以记为：

$$\min_{M,U,H,C}\|S-MU^T\|_F^2+\alpha\|H-UC^T\|_F^2-\beta tr(H^T\mathrm{BH})$$

可以看出目标函数由三个部分组成，分别是$\min_{M,U}\|S-MU^T\|_F^2$，$\min_{U,H,C}\alpha\|H-UC^T\|_F^2$)和$\max_H\beta tr(H^T\mathrm{BH})$。这三个部分分别表示了将相似矩阵 S 分解为 M 和 U 矩阵的效果最好，将社区指示矩阵 H 分解为 U 和 C 矩阵的效果最好，以及划分后的模块度最高三个期望的条件。最后我们只要迭代更新这四个变量就可以了，前三个变量更新的公式是由目标函数分别对每一个变量求偏导得到的：

$$\mathrm{M}\leftarrow\mathrm{M}\odot\frac{SU}{MU^TU}$$

$$\mathrm{U}\leftarrow\mathrm{U}\odot\frac{S^TM+\alpha HC}{U(M^TM+\alpha C^TC)}$$

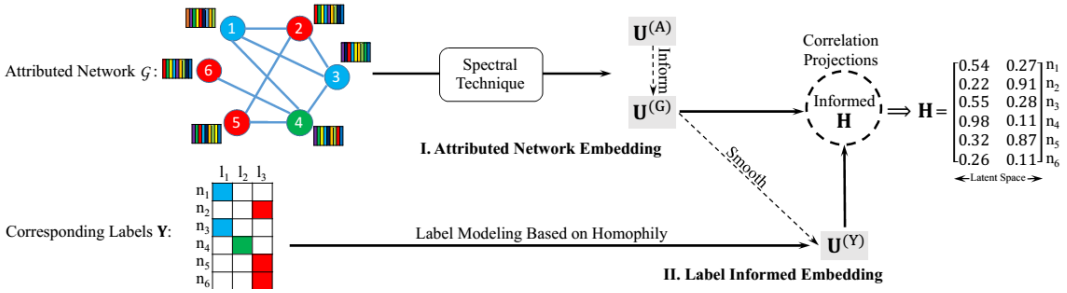
$$\mathrm{C}\leftarrow\mathrm{C}\odot\frac{H^TU}{CU^TU}$$

$$\mathrm{H}\leftarrow\mathrm{H}\odot\sqrt{\frac{-2\beta B_1H+\sqrt{\Delta}}{8\lambda HH^TH}}$$

最后更新至目标收敛即可认为达到了 network embedding 的目标。矩阵 U 就是得到的矩阵。

## LANE

LANE 算法的主要目的是将矩阵的信息（邻接矩阵，特征矩阵和类标）通过谱方法进行特征的提取，最后再共同融合到矩阵 H 中，作为 network embedding 的结果。




我们以特征矩阵 A 为例，首先通过计算余弦相似度得到矩阵$S^{(A)}$。

$S^{(A)} = \frac{N_i N_j}{\|N_i\|\|N_i\|}$，其中$N_i$是 A 的第 i 行。

然后构建拉普拉斯矩阵，目标是通过谱方法实现降维。

$$L^{(A)} = D^{(A)^{-1/2}} S^{(A)} D^{(A)^{-1/2}}$$

$$\left(\alpha_1 L^{(A)} + \alpha_1 U^{(G)} U^{(G)^T} + HH^T\right) U^{(A)} = \lambda_1 U^{(A)}$$

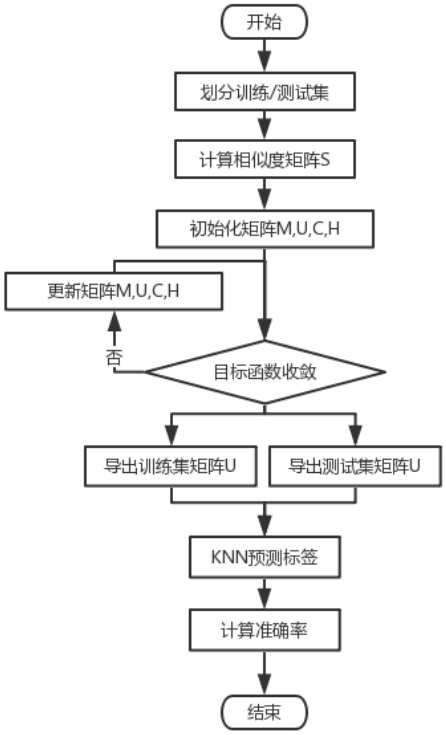
最后通过优化目标函数，

$$\max_{U(\cdot),H} J = (J_G + \alpha_1 J_A + \alpha_1 \rho_1) + \alpha_2 J_Y + J_{corr}$$

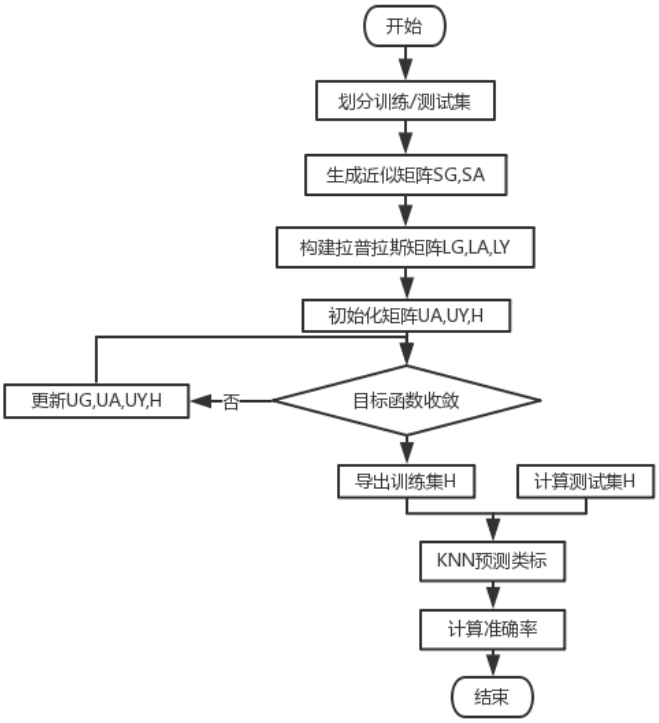
就可以实现将网络的 G，A，Y 矩阵中的特征都提取入矩阵 H 的目的

## 2. 算法流程

### M_NMF




### LANE



## 3. 关键代码截图

## M_NMF

划分训练集与测试集;

```
[n, ~] = size(A);
n_train = n*0.6;
A_train = A(1:n_train, 1:n_train);
A_test = A(n_train+1:n, n_train+1:n);
L_train = L(1:n_train);
L_test_real = L(n_train+1:n);
```

生成特征矩阵 S;

```
S1 = A;
S2 = zeros(n, n);
for i=1:n
    for j=1:n
        S2(i, j) = (S1(i, :) * (S1(j, :))') / (norm(S1(i, :)) * norm(S1(j, :)));
    end
end
S = S1+5 .* S2;
```

目标函数;

```
function aim = AIM(S, M, U, H, C, B, alpha, beta)

aim = (sum_fro(S-M*(U')))^2 + alpha * (sum_fro(H-U*(C')))^2 - beta * trace((H')*B*H);

end
```

迭代更新的过程;

```
while(abs(aim0 - aim1) > 0.0000000001)
    M0 = M .* ((S*U)./(M*(U')*U));
    U0 = U .* (((S')*M+alpha*H*C)./(U*((M')*M+alpha*(C')*C)));
    C0 = C .*(((H')*U)./(C*(U')*U));
    delta = (2*beta*(B1*H)) .* (2*beta*(B1*H)) + 16*lambda*(H*(H')*H) .* (2*beta*A*H+2*alpha*U*(C')+(4*lambda-2*alpha)*H);
    H0 = H .* sqrt((-2*beta*B1*H+sqrt(delta))./(8*lambda*H*(H')*H));
    M = M0;
    U = U0;
    C = C0;
    H = H0;
    aim0 = aim1;
    aim1 = AIM(S, M, U, H, C, B, alpha, beta);
    disp(aim1);
    iter = iter +1;
end
```

预测测试集，导入 KNN 计算准确率;

```
U_train = M_NMF(A_train, k);
U_test = M_NMF(A_test, k);
L_test = KNN(U_train, L_train, U_test, k);
ac = classificationACC(L_test_real, L_test);
```

# LANE

划分训练集测试集;

```
n_train = n*0.6;
G_train = G(1:n_train,1:n_train);
A_train = A(1:n_train,:);
A_test = A(n_train+1:n,:);
Y_train = Y(1:n_train);
Y_test_real = Y(n_train+1:n);
```

构建特征矩阵，注意分母为零时的处理;

```
SG = zeros(n,n);
for i=1:n
    for j=1:n
        if(norm(G(i,:))*norm(G(j,:))==0)
            SG(i,j) = 0;
        else
            SG(i,j) = (dot(G(i,:), G(j,:)))/(norm(G(i,:))*norm(G(j,:)));
        end
    end
end
SA = zeros(n,n);
for i=1:n
    for j=1:n
        if(norm(A(i,:))*norm(A(j,:))==0)
            SA(i,j) = 0;
        else
            SA(i,j) = (dot(A(i,:), A(j,:)))/(norm(A(i,:))*norm(A(j,:)));
        end
    end
end
SY = zeros(n,n);
YY = Y*(Y');
for i=1:n
    for j=1:n
        if(norm(YY(i,:))*norm(YY(j,:))==0)
            SY(i,j) = 0;
        else
            SY(i,j) = (dot(YY(i,:), YY(j,:)))/(norm(YY(i,:))*norm(YY(j,:)));
        end
     end
end
```

构建拉普拉斯矩阵，为了防止矩阵的-1/2 无意义，构建 D 矩阵时加 0.1;

```
DG = zeros(n,n);
for i=1:n
    DG(i,i) = sum(SG(i,:))+0.1;
end
DA = zeros(n,n);
for i=1:n
    DA(i,i) = sum(SA(i,:))+0.1;
end
DY = zeros(n,n);
for i=1:n
    DY(i,i) = sum(SY(i,:))+0.1;
end
LG = (DG ^(-1/2)) * SG * (DG ^(-1/2));
LA = (DA ^(-1/2)) * SA * (DA ^(-1/2));
LY = (DY ^(-1/2)) * SY * (DY ^(-1/2));
```

目标函数;

```
function J = calc_J(alpha1,alpha2,UG,UA,UY,LG,LA,LY,H)
JG = trace(UG'*LG*UG);
JA = trace(UA'*LA*UA);
JY = trace(UY'*LY*UY);
rou1 = trace(UA'*UG*(UG')*UA);
rou2 = trace(UG'*H*(H')*UG);
rou3 = trace(UA'*H*(H')*UA);
rou4 = trace(UY'*H*(H')*UY);
Jcorr = rou2 + rou3 + rou4;
J=JG+alpha1*JA+alpha1*rou1+alpha2*JY+Jcorr;
end
```

迭代优化;

```
while(flag == 1)
    M0 = LG + alpha1*UA*(UA') + alpha2*UY*(UY') + H*(H');
    [UG0,~] = eigs(M0,d);
    M1 = alpha1*LA + alpha1*UG*(UG') + H*(H');
    [UA0,~] = eigs(M1,d);
    M2 = alpha2*LY + alpha2*UG*(UG') + H*(H');
    [UY0,~] = eigs(M2,d);
    M3 = UG*(UG') + UA*(UA') + UY*(UY');
    [H0,~] = eigs(M3,d);
    UG = UG0;
    UA = UA0;
    UY = UY0;
    H = H0;
    J0 = J1;
    J1 = calc_J(alpha1,alpha2,UG,UA,UY,LG,LA,LY,H);
    iter=iter+1;
    if(abs(J1-J0) < epsion)
        flag = 0;
    end
end
```

预测训练集结果，计算测试集结果，然后导入 KNN。最后计算准确率；

```
H_train = LANE(d, G_train, A_train, Y_train);
G1 = G(1:n_train, :);
G2 = G(n_train+1:n, :);
H_test = G2*pinv(pinv(H_train)*G1)+delta*A_test*pinv(pinv(H_train)*A_train);
Y_test = KNN(H_train, Y_train, H_test, k);
ac = classificationACC(Y_test_real, Y_test);
```

# 三、 实验结果及分析

**1. 评测指标展示即分析**

## M_NMF

| 数据集 | Acc |
|---|---|
| cornell | 0.4359 |
| texas | 0.5676 |
| washington | 0.0543 |
| wisconsin | 0.4811 |

## LANE

| 数据集 | Acc |
|---|---|
| cornell | 0.5641 |
| texas | 0.6757 |
| washington | 0.5761 |
| wisconsin | 0.6981 |

综合来说两个算法的表现都比较一般，当然考虑到我们强行使用 network embedding 算法来进行分类的任务，最好的表现自然会受到最后 KNN 的影响，因此出现 0.0543 之类的数据自然也不是非常奇怪。但是我们也可以从实验的初步结果看出相较于 M_NMF 算法，LANE 算法更加趋向于稳定，在后接 KNN 分类器的情况下仍能够保持一个比较比较稳定的表现。并且 LANE 的总体准确率也是高于 M_NMF 的。相比于只是用了相似矩阵 S 的 M_NMF 算法，LANE 算法用到了网络的更多信息，因此这个现象也在意料之内。