本科生实验报告

（2017-2018 学年秋季学期）

一、 实验题目

Network Embedding

二、 算法解析

1. M-NMF

M-NMF 是一种结合了一阶相似度、二阶相似度和社区结构的网络嵌入模型，其中三方面分别用如下方式来表示：

一阶相似度：邻接矩阵，表示节点的邻居节点

二阶相似度：用于表示有共同邻居节点的节点的相似性。

$$S_{ij}^{(2)} = \frac{\mathcal{N}_i \mathcal{N}_j}{||\mathcal{N}_i|| ||\mathcal{N}_j||}$$

社区结构：通过 modularity 来刻画，modularity Q 定义如下：

$$Q = \frac{1}{2e}\sum_{ij}[A_{ij} - \frac{k_i k_j}{2e}]\delta(h_i, h_j) = Tr(H^T BH)$$

相似度矩阵通过一阶相似度和二阶相似度加权得到，$S = S^{(1)} + \eta S^{(2)}$，实验中取 $\eta = 5$.

将以上三部分结合起来最终的目标函数为：

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

$$s.t., M \geq 0, U \geq 0, H \geq 0,\ C \geq 0,\ \mathrm{tr}(H^T H) = \mathrm{n}$$

矩阵 H 是社区指示矩阵，各行代表的是各节点所属社区；U 为节点特征矩阵；C 为社区特征矩阵。

算法流程：

（1）计算相似度矩阵： $S = S^{(1)} + \eta S^{(2)}$

（2）迭代更新 M,U,C,H 直至收敛

为 4 个变量设置随机初始值，根据更新公式进行迭代，直至目标函数小于某个阈值停止。

更新公式如下：

$$M \leftarrow M \bullet \frac{SU}{MU^T U}$$

$$U \leftarrow U \bullet \frac{S^T M + \alpha HC}{U(M^T M + \alpha C^T C)}$$

$$C \leftarrow C \bullet \frac{H^T U}{CU^T U}$$

$$H \leftarrow H \bullet \sqrt{\frac{-2\beta B_1 H + \sqrt{\Delta}}{8\lambda HH^T H}}$$

（3）使用矩阵 U 结合分类算法进行分类

划分训练集、验证集，80%训练，20%验证。使用矩阵 U 以及 label 进行训练、测试。实验中采用了 SVM 算法。

2. LANE

LANE 利用三种信息：邻接矩阵 G、节点属性矩阵 A、节点类标信息 Y。

LANE 是一种标签信息属性网络嵌入方法，它可以对属性网络空间和标签信息空间中的节点近似值以及标签信息空间进行建模并联合嵌入到一个统一的低维表示中。

主要有两个步骤：第一，将网络结构和属性信息中的节点近似值映射为两个潜在的表示形式。U(G)和 U(A)，然后通过提取 U(G)和 U(A)的相关性，将 U(A)合并到 U(G)中。第二，利用学习到的联合接近来平滑标签信息，并将它们一致嵌入到其他表示 U(Y)，然后将所有学习到的隐表示投影到一个统一的嵌入表示 H 中。

目标函数如下：

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

算法流程：

(1) 划分训练集与验证集

对邻接矩阵 G、节点属性矩阵 A、节点类标信息 Y 三个矩阵分别进行划分。80%训练，20%验证。产生随机顺序序列，序列所在的前 80%为训练节点。

(2) LANE 计算 H_train

a. 对 G、A、Y 分别进行计算得到对应的余弦相似度矩阵 S(G)、S(A)、S(Y)；

b. 求对角矩阵 D(G)、D(A)、D(Y),根据 S(G)、S(A)、S(Y)每行的和；

c. 求拉普拉斯矩阵 L(G)、L(A)、L(Y)，根据 $L = D^{-1/2} S D^{-1/2}$；

d. 迭代更新 U(G)、U(A)、U(Y)、H，直至目标函数改变小于某一阈值，之后得到的为 H_train.

通过如下计算如下四个方程的最大的 d 个特征值对应的特征向量可以得到对应的更新矩阵，d 为所降维数，人工设定。

$$(\mathcal{L}^{(G)} + \alpha_1 \mathbf{U}^{(A)}\mathbf{U}^{(A)^T} + \alpha_2 \mathbf{U}^{(Y)}\mathbf{U}^{(Y)^T} + \mathbf{H}\mathbf{H}^T)\mathbf{U}^{(G)} = \boldsymbol{\lambda}_1 \mathbf{U}^{(G)}$$

$$(\alpha_1 \mathcal{L}^{(A)} + \alpha_1 \mathbf{U}^{(G)}\mathbf{U}^{(G)^T} + \mathbf{H}\mathbf{H}^T)\mathbf{U}^{(A)} = \boldsymbol{\lambda}_2 \mathbf{U}^{(A)},$$

$$(\alpha_2 \mathcal{L}^{(YY)} + \alpha_2 \mathbf{U}^{(G)}\mathbf{U}^{(G)^T} + \mathbf{H}\mathbf{H}^T)\mathbf{U}^{(Y)} = \boldsymbol{\lambda}_3 \mathbf{U}^{(Y)},$$

$$(\mathbf{U}^{(G)}\mathbf{U}^{(G)^T} + \mathbf{U}^{(A)}\mathbf{U}^{(A)^T} + \mathbf{U}^{(Y)}\mathbf{U}^{(Y)^T})\mathbf{H} = \boldsymbol{\lambda}_4 \mathbf{H}.$$

(3) 计算 H_test

$$\mathbf{H}_{test} = \mathbf{G}_2((\mathbf{H}_{train})^{\dagger}\mathbf{G}_1)^{\dagger} + \delta \mathbf{A}_{test}((\mathbf{H}_{train})^{\dagger}\mathbf{A}_{train})^{\dagger}$$

(4) 使用分类器对 H 进行训练测试

3. LINE

以下介绍基于二阶相似度的 LINE 算法。

算法思想为通过 embedding 得到的结果矩阵 U 产生的概率分布（给定节点 Vi,出现节点 Vj 的条件概率）应该尽可能地接近真实的分布，用 KL 散度测量它这两个分布是不相关性（距离），并最小化这个距离。

给定节点 Vi,出现节点 Vj 的条件概率：

$$p_2(v_j|v_i) = \frac{{u'_j}^T \cdot u_i}{\sum_{k=1}^{|V|} \exp({u'_k}^T \cdot u_i)}$$

目标函数：

$$O_2 = -\sum_{(i,j)\in E} w_{ij} \log p_2(v_j|v_i)$$

在进行优化时采用负采样，只选取几个不相似的点进行计算，根据节点的度数来进行负采样，节点的度数越大，被采样的概率越高。

算法流程：

(1) 为节点度数创建线段列表，用于进行负采样

(2) 随机初始化 U2、U',然后进行迭代更新

U1/U2 为 embedding 结果矩阵；U'为节点做诶邻居的表达，即二阶相似度。

(3) 采取随机梯度下降法，迭代每一步中，选择一定数量的边进行更新；

选取的每一条边中其中一个作为源节点，另一个为目标节点。通过负采样选择不相似的

点来进行负点的的更新，更新公式如下：

$$u_i = u_i - \rho\frac{\partial O_2}{\partial u_i} \quad u'_j = u'_j - \rho\frac{\partial O_2}{\partial u'_j} \quad u'_k = u'_k - \rho\frac{\partial O_2}{\partial u'_k}$$

三、 代码实现

1. M-NMF

（1）计算相似度矩阵

```
%求相似度矩阵
[n,~]=size(A);
S=zeros(n,n);
for i=1:n
    for j=1:n
        S(i,j)=A(i,:)*A(j,:)'/(norm(A(i,:)).*norm(A(j,:)));
    end
end
S=A+5*S;%相似度矩阵
```

（2）迭代更新 M,U,C,H 直至收敛

```
k=5;%社区数
m=100;%每个点表示的维数
U=rand(n,m);%表示矩阵
M=rand(n,m);%基矩阵
C=rand(k,m);%社区表示矩阵
H=rand(n,k);%对第i行，若第j列为1则属于该社区，其他列为0
maxiter=30; %最大迭代次数
a=1;b=5;la=10.^9;
for i=1:maxiter
    M=M.*((S*U)./(M*(U'*U)));
    U=U.*((S'*M+a*H*C)./(U*(M'*M+a.*C'*C)));
    C=C.*((H'*U)./(C*(U'*U)));
    Del=2*b*(B1*H).*(2*b*(B1*H))+16*la*((H*H')*H).*(2*b*A*H+2*a*U*C'+(4*la-2*a)*H);
    H=H.*sqrt((-2*b*B1*H+sqrt(Del))./(8*la*(H*H')*H));
    Obj(i)=norm(S-M*U','fro').^2+a*norm(H-U*C','fro').^2-b*trace(H'*B*H);
end
```

（3）划分数据集

```
%划分数据集
tem=randperm(n);
trainNum=round(0.8*n);
testNum=n-trainNum;
train_M=zeros(trainNum,m);train_label=zeros(trainNum,1);
test_M=zeros(testNum,m);test_label=zeros(testNum,1);
for i=1:trainNum
    train_M(i,:)=U(tem(i),:);
    train_label(i)=label(tem(i));
end
for i=1:testNum
    test_M(i,:)=U(tem(i+trainNum),:);
    test_label(i)=label(tem(i+trainNum));
end
```

（4）SVM 分类

```
t=templateSVM('Standardize',true);
model=fitcecoc( train_M,train_label,'Learners',t);
predicted_label=predict(model,test_M);
classificationACC(test_label,predicted_label)
```

2. LANE

(1) 划分训练集与验证集

对 A 和 Y 只需要取出对应行即可；对矩阵 G 在行列分别按照随机序列顺序排序之后，G_train 取出得到方阵，因为在训练的时候测试集节点的信息是不可知的，而 G_test 取测试节点到所有节点的信息。

```
tem=randperm(n);
A=A(tem,:);A=A(:,tem);%对A随机排序
F=F(tem,:);label=label(tem,:);
trainNum=round(0.8*n);
testNum=n-trainNum;
train_G=A(1:trainNum,1:trainNum);G1=A(1:trainNum,:);
test_G=A(trainNum+1:n,:);G2=A(trainNum+1:n,:);
train_A=F(1:trainNum,:);
test_A=F(trainNum+1:n,:);
train_label=label(1:trainNum);test_label=label(trainNum+1:n);
```

(2) 迭代更新 U(G)、U（H）、U（Y）、H，计算目标函数

```
for t=1:maxiter
    M_G=L_G+a1*U_A*U_A'+a2*U_Y*U_Y'+H*H';
    [U_G,~]=eigs(M_G,d);
    M_A=a1*L_A+a1*U_G*U_G'+H*H';
    [U_A,~]=eigs(M_A,d);
    M_Y=a2*L_YY+a2*U_G*U_G'+H*H';
    [U_Y,~]=eigs(M_Y,d);
    M_H=U_G*U_G'+U_A*U_A'+U_Y*U_Y';
    [H,~]=eigs(M_H,d);
    J_A=trace(U_A'*L_A*U_A);
    J_G=trace(U_G'*L_G*U_G);
    J_Y=trace(U_Y'*(L_YY+U_G*U_G')*U_Y);
    J_corr=trace(U_G'*H*H'*U_G)+trace(U_A'*H*H'*U_A)+trace(U_Y'*H*H'*U_Y);
    p1=trace(U_A'*U_G*U_G'*U_A);
    J(t)=(J_G+a1*J_A+a1*p1)+a2*J_Y+J_corr;
end
```

(1) 为节点度数创建线段列表，用于进行负采样

```
%生成采样度数线段
D=sum(A);%每个点的度数
Dlist=zeros(n,2);%每个点的起点终点
Dlist(1,:)=[0 D(1)];
for i=2:n
    Dlist(i,1)=Dlist(i-1,2) ;
    Dlist(i,2)=Dlist(i,1)+D(i);
end
```

(2) 生成负采样点

```
%负采样点
sample=randperm(len,K);%产生K个随机负采样点
NegSam=zeros(K,1);%获取采样点
for i=1:K
    for j=1:n
        if sample(i)>=Dlist(j,1) && sample(i)<= Dlist(j,2)
            NegSam(i)=j;
            break;
        end
    end
end
```

(3) 异步随机梯度下降选取一定数量的边进行迭代更新

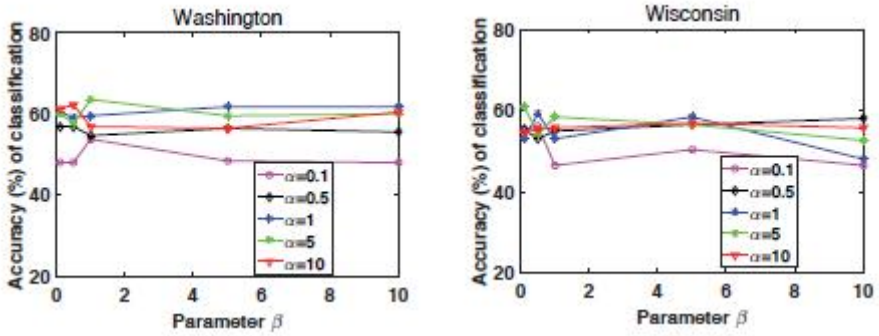
```
tem=randperm(EdgeNum,batchNum);
for i=1:batchNum
    vi=X(tem(i)); vj=Y(tem(i));
    Sum=0;
    for j=1:K
        k=NegSam(j);
        delUk = A(vi,vj)*U2(vi,:)*sigmoid(U(k,:)'*U2(vi,:));
        Sum = Sum + U(k,:)*sigmoid(U(k,:)'*U2(vi,:));
    end
    delUi=-A(vi,vj)*(U(vj,:)*(1-sigmoid(U(vj,:)'*U2(vi,:)))-Sum);
    delUj=-A(vi,vj)*U2(vi,:)*(1-sigmoid(U(vj,:)'*U2(vi,:)));
    U2(vi,:)=U2(vi,:)-rho*delUi;
    U(vj,:)=U(vj,:)-rho*delUj;
    for j=1:K%更新每一个负点
        k=NegSam(j);
        U(k,:)=U(k,:)-rho*delUk(:,j);
    end
end
```

四、 效果分析

1. M-NMF

参数选取：

可变参数，包括 a 和 b, 根据“Community Preserving Network Embedding”论文中的实验结果，采用 a=1,b=5 进行实验。



采用 SVM 进行分类，80%训练，ACC 如下表：

Datasets	M_NMF
cornell	0.4359
texas	0.7297
washington	0.6304
wisconsin	0.5849

2. LANE

实验中的两个可变参数为α_1,α_2，根据论文中的 a 值选取，在α_1,α_2都比较大的时候能

够获得比较好的效果。实验采用，`d=100;a1=10;a2=100;`

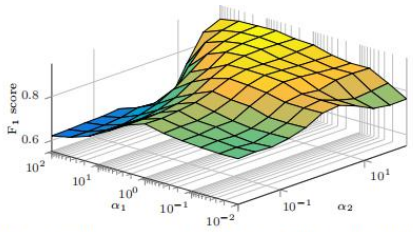


Figure 3: Performance of LANE on Flickr with different parameters α_1 and α_2.

使用 SVM 进行分类，80%训练，ACC 如下表：

Datasets	LANE
cornell	0.68803
texas	0.6108
washington	0.6087
wisconsin	0.7736

3. LINE

使用 SVM 进行分类，80%训练，ACC 如下表：

Datasets	LINE
cornell	0.4359
texas	0.5028
washington	0.5312
wisconsin	0.4603

对比以上三种算法,LANE表现比较好且稳定。