**import** numpy **as** np  
**import** math  
**import** random  
**from** sklearn.metrics **import** roc\_curve  
**from** sklearn.metrics **import** roc\_auc\_score  
**import** pandas **as** pd  
**def** getSimilarMatrix(IP, γ\_):  
 dimensional = IP.shape[0]  
 sd = np.zeros(dimensional)  
 K = np.zeros((dimensional, dimensional))  
 **for** i **in** range(dimensional):  
 sd[i] = np.linalg.norm(IP[i]) \*\* 2  
 gamad = γ\_ \* dimensional / np.sum(sd.transpose())  
 **for** i **in** range(dimensional):  
 **for** j **in** range(dimensional):  
 K[i][j] = math.exp(-gamad \* (np.linalg.norm(IP[i] - IP[j])) \*\* 2)  
 **return** K  
**def** Kfoldcrossclassify(sample, K, fun=**"cv3"**):  
 r = []  
 **if** fun != **"cv3"**:  
 m = np.mat(sample)  
 **if** fun == **"cv1"**:  
 t = 0  
 **else**:  
 t = 1  
 mt = Kfoldcrossclassify(np.array(range(np.max(m[:, t]) + 1)), K)  
 r = [[j **for** j **in** sample **if** j[t] **in** mt[i]] **for** i **in** range(K)]  
 **return** r  
 l = sample.shape[0]  
 t = sample.copy()  
 n = math.floor(l / K)  
 retain = l - n \* K  
 **for** i **in** range(K - 1):  
 nt = n  
 e = len(t)  
 *# if e % n and e % K:* **if** retain > i:  
 nt += 1  
 a = random.sample(range(e), nt)  
 r.append([t[i] **for** i **in** a])  
 t = [t[i] **for** i **in** range(e) **if** (i **not in** a)]  
 r.append(t)  
 **return** r  
**def** svt(Y, x):  
 S, v, D = np.linalg.svd(Y)  
 D = D.T  
 V = np.diag(v)  
 V\_row, V\_col = V.shape  
 x = x \* np.ones(v.size)  
 v\_new = np.zeros(v.size)  
 noneZero = v > x  
 v\_new[noneZero] = v[noneZero] - x[noneZero]  
 **if** V\_row < V\_col:  
 E = S @ np.hstack((np.diag(v\_new), np.zeros((V\_row, V\_col - V\_row)))) @ D.T  
 **else**:  
 E = S @ np.vstack((np.diag(v\_new), np.zeros((V\_row - V\_col, V\_col)))) @ D.T  
 **return** E  
**def** BNNR(alpha, beta, T, trIndex, tol1, tol2, maxtier, a, b):  
 A = T.copy()  
 W = A.copy()  
 B = A.copy()  
 i = 1  
 stop1 = 1  
 stop2 = 1  
 **while** (stop1 > tol1 **or** stop2 > tol2):  
 tran = (1 / beta) \* (B + alpha \* (T \* trIndex)) + A  
 W = tran - (alpha / (alpha + beta)) \* (tran \* trIndex)  
 W[W < a] = a  
 W[W > b] = b  
 A\_1 = svt(W - 1 / beta \* B, 1 / beta)  
 B = B + beta \* (A\_1 - W)  
 stop1\_0 = stop1  
 stop1 = np.linalg.norm(A\_1 - A) / np.linalg.norm(A)  
 stop2 = abs(stop1 - stop1\_0) / max(1, abs(stop1\_0))  
 A = A\_1  
 i = i + 1  
 **if** i < maxiter:  
 \_iter = i - 1  
 **else**:  
 \_iter = maxiter  
 print(**'reach maximum iteration~~do not converge!!!'**)  
 **break** T\_recovery = W  
 **return** T\_recovery, \_iter  
a\_filename = **"../dataset/data1/sm\_v.xlsx"**dd\_filename = **"../dataset/data1/sm\_s.xlsx"**vv\_filename = **"../dataset/data1/v\_s.xlsx"**Wdv = pd.read\_excel(a\_filename, header=**None**).to\_numpy()  
Wvv = pd.read\_excel(vv\_filename, header=**None**).to\_numpy()  
Wdd = pd.read\_excel(dd\_filename, header=**None**).to\_numpy()  
Wdv = Wdv.T  
dn, dr = Wdv.shape  
maxiter = 400  
tol1 = 2 \* 1e-3  
tol2 = 1 \* 1e-5  
a = np.array([(i, j) **for** i **in** range(dn) **for** j **in** range(dr) **if** Wdv[i, j]])  
b = np.array([(i, j) **for** i **in** range(dn) **for** j **in** range(dr) **if** Wdv[i, j] == 0])  
result\_list = []  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 max\_auc = 0  
 t = **None  
 for** alpha **in** [0.1, 1, 10, 100]:  
 **for** beta **in** [0.1, 1, 10, 100]:  
 **for** w **in** np.arange(0, 1, 0.1):  
 **for** γ\_ **in** np.arange(0.5, 3, 0.5):  
 AUCS = 0  
 z = 10  
 **for** h **in** range(z):  
 f = Kfoldcrossclassify(a, 5, fun=**"cv3"**)  
 AUCs = 0  
 **for** i **in** range(5):  
 test\_sample = np.array(f[i])  
 negative\_sample = np.array(b)  
 Wdv\_ = Wdv.copy()  
 Wdv\_[test\_sample[:, 0], test\_sample[:, 1]] = 0  
 GV = getSimilarMatrix(Wdv\_, γ\_)  
 GD = getSimilarMatrix(Wdv\_.T, γ\_)  
 SV = w \* GV + (1 - w) \* Wvv  
 SD = w \* GD + (1 - w) \* Wdd  
 T = np.vstack((np.hstack((SD, Wdv\_.T)), np.hstack((Wdv\_, SV))))  
 t1, t2 = T.shape  
 trIndex = T != 0  
 WW, \_iter = BNNR(alpha, beta, T, trIndex, tol1, tol2, maxiter, 0, 1)  
 M\_recovery = WW[t1 - dn: t1, : dr]  
  
 test\_sample\_number = test\_sample.shape[0]  
 negative\_sample\_number = negative\_sample.shape[0]  
 label = test\_sample\_number \* [1] + negative\_sample\_number \* [0]  
 label = np.array(label)  
 sample = np.vstack((test\_sample, negative\_sample))  
 score = M\_recovery[sample[:, 0], sample[:, 1]]  
 fpr, tpr, threshold = roc\_curve(label, score)  
 auct = roc\_auc\_score(label, score)  
 *# auct = auc(fpr, tpr)  
 # print(auct)* AUCs += auct  
 AUC\_m = AUCs/5  
 *# print(AUC\_m)* AUCS += AUC\_m  
 AUC\_mean = AUCS/z  
 result\_list.append({**"AUC"**: AUC\_mean, **"args"**: {**"alpha"**: alpha, **"beta"**: beta, **'w'**: w, **'γ\_'**: γ\_}})  
pd.DataFrame(result\_list).to\_excel(**"grid\_test\_cv3\_1.xlsx"**)