**import** numpy **as** np  
**import** pandas **as** pd  
**import** math  
**import** random  
**from** sklearn.metrics **import** roc\_curve  
**from** sklearn.metrics **import** roc\_auc\_score  
**from** sklearn.metrics **import** precision\_recall\_curve  
**from** sklearn.metrics **import** average\_precision\_score  
**from** sklearn.metrics **import** f1\_score, accuracy\_score  
**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/data3/sm\_v.xlsx"**dd\_filename = **"../dataset/data3/sm\_s.xlsx"**vv\_filename = **"../dataset/data3/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  
indicator\_t = []  
maxiter = 400  
alpha = 1  
beta = 0.1  
w = 0.5  
γ\_ = 0.5  
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])  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 note = []  
 AUCS, AUPRS, PrecisionS, RecallS, F1\_scoreS, AccuracyS, SensitivityS, SpecificityS = 0, 0, 0, 0, 0, 0, 0, 0  
 z = 100  
 **for** h **in** range(z):  
 f = Kfoldcrossclassify(a, 5, fun=**"cv3"**)  
 AUCs, AUPRs, precisions, recalls, f1\_scores, accuracys, sensitivitys, specificitys = 0, 0, 0, 0, 0, 0, 0, 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)   
 note.append((auct, fpr, tpr))  
 aupr = average\_precision\_score(label, score)  
 precision, recall, threshold2 = precision\_recall\_curve(label, score)  
 acc\_t = sum([accuracy\_score(label, score >= thre) **for** thre **in** threshold])/threshold.size  
 precision\_t = precision.mean()  
 recall\_t = recall.mean()  
 f1\_score\_t = sum([f1\_score(label, score >= thre) **for** thre **in** threshold2])/threshold2.size  
 AUCs += auct  
 AUPRs += aupr  
 accuracys += acc\_t  
 precisions += precision\_t  
 recalls += recall\_t  
 f1\_scores += f1\_score\_t  
 specificity\_t = 0  
 **for** j **in** range(threshold.size):  
 TP, TN, FP, FN = (0, 0, 0, 0)  
 threshold\_value = threshold[j]  
 **for** k **in** range(score.size):  
 predicted\_value = score[k]  
 **if** predicted\_value >= threshold\_value:  
 **if** label[k]:  
 TP += 1  
 **else**:  
 FP += 1  
 **else**:  
 **if** label[k]:  
 FN += 1  
 **else**:  
 TN += 1  
 specificity\_t += TN / (TN + FP)  
 specificitys += specificity\_t / (threshold.size)  
 AUC\_m = AUCs/5  
 AUPR\_m = AUPRs / 5  
 acc\_m = accuracys / 5  
 precision\_m = precisions / 5  
 recall\_m = recalls / 5  
 f1\_score\_m = f1\_scores / 5  
 specificity\_m = specificitys / 5  
 AUCS += AUC\_m  
 AUPRS += AUPR\_m  
 AccuracyS += acc\_m  
 PrecisionS += precision\_m  
 RecallS += recall\_m  
 F1\_scoreS += f1\_score\_m  
 SpecificityS += specificity\_m  
 AUC\_mean = AUCS/z  
 AUPR\_mean = AUPRS/z  
 Accuracy\_mean = AccuracyS / z  
 Precision\_mean = PrecisionS / z  
 Recall\_mean = RecallS / z  
 F1\_score\_mean = F1\_scoreS / z  
 Sensitivity\_mean = Recall\_mean  
 Specificity\_mean = SpecificityS / z  
 indicator\_t.append({**"AUC"**: AUC\_mean, **'AUPR'**: AUPR\_mean, **'Accuracy'**: Accuracy\_mean, **'Precision'**: Precision\_mean,  
 **'Recall'**: Recall\_mean, **'F1\_score'**: F1\_score\_mean, **'Specificity'**: Specificity\_mean})  
 mm = 0  
 kk = 0  
 **for** x **in** range(len(note)):  
 mm += note[x][0]  
 nn = mm/len(note)  
 t\_auc = np.inf  
 dd = **None  
 for** y **in** range(len(note)):  
 kk = abs(note[y][0]-nn)  
 **if** kk < t\_auc:  
 t\_auc = kk  
 dd = (note[y][1],note[y][2])  
 fpr, tpr = dd  
 **with** open(**"fpr\_cv3\_3\_GIPK.txt"**, **"w"**) **as** fp:  
 **for** i **in** fpr:  
 fp.write(str(i)+**' '**)  
 **with** open(**'tpr\_cv3\_3\_GIPK.txt'**, **'w'**) **as** fp:  
 **for** i **in** tpr:  
 fp.write(str(i)+**' '**)  
pd.DataFrame(indicator\_t).to\_excel(**"final indicators on cv3\_3.xlsx"**)