拓扑代码数据

第1种 P-P-D（已手动验证）

```cpp
#include<iostream>

using namespace std;
int node;
int count;
int nodes[5000];
struct xulie {
    int a, b;
} xulie[500749];
struct jibing {
    char c1, c2;
} dis[155];
struct guanxi {
    int a;
    char c1;
} gx[5002];

int main() {
    freopen("node.txt", "r", stdin);
    int s = 0, s1 = 0, s2 = 0, s3 = 0;
    while (cin >> node) {
        nodes[s++] = node;
    }
    cin.clear();
    freopen("xuliesim.txt", "r", stdin);
    int i, j;
    while (cin >> i >> j) {
        xulie[s1].a = i;
        xulie[s1].b = j;
        s1++;

    }
    cin.clear();
    char c1, c2;
    freopen("bing.txt", "r", stdin);
    while (cin >> c1 >> c2) {
        dis[s2].c1 = c1;
        dis[s2].c2 = c2;
        s2++;
    }
    cin.clear();
    freopen("bingxu.txt", "r", stdin);
    int shu;
    char fu;
    while (cin >> shu >> fu) {
        gx[s3].a = shu;
        gx[s3].c1 = fu;
        s3++;
    }
    freopen("out.txt", "w", stdout);
    for (i = 0; i < s; i++) {
        count=0;
        //求第5种疾病代号为e
        //P-P-D开始求解 对每种piRNA开始终点是e
        //s是结点个数 s1是序列个数 s2是疾病个数 s3是病序
        for (j = 0; j < s1; j++) {
            //P-P
            if (xulie[j].a == nodes[i] || xulie[j].b == nodes[i]) {
                {
                    int zd;
                    if (xulie[j].a == nodes[i]) {
                        zd = xulie[j].b;
                    } else {
                        zd = xulie[j].a;
                    }
                    //P-D
                    for (int k = 0; k < s3; k++) {
                        if (gx[k].a == zd) {
                            if (gx[k].c1 == 'e') {
                                count++;
                            }
                        }
                    }


                }


            }
        }
    cout<<count<<endl;

    }

    return 0;
}
```

第2种 P-D-P-D（已手动验证）

```
#include<iostream>

using namespace std;
int node;
int count;
int nodes[5000];
struct xulie {
    int a, b;
} xulie[500749];
struct jibing {
    char c1, c2;
} dis[155];
struct guanxi {
    int a;
    char c1;
} gx[5002];

int main() {
    freopen("node.txt", "r", stdin);
    int s = 0, s1 = 0, s2 = 0, s3 = 0;
    while (cin >> node) {
        nodes[s++] = node;
    }
    cin.clear();
    freopen("xuliesim.txt", "r", stdin);
    int i, j;
    while (cin >> i >> j) {
        xulie[s1].a = i;
        xulie[s1].b = j;
        s1++;

    }
    cin.clear();
    char c1, c2;
    freopen("bing.txt", "r", stdin);
    while (cin >> c1 >> c2) {
        dis[s2].c1 = c1;
        dis[s2].c2 = c2;
        s2++;
    }
    cin.clear();
    freopen("bingxu.txt", "r", stdin);
    int shu;
    char fu;
    while (cin >> shu >> fu) {
        gx[s3].a = shu;
        gx[s3].c1 = fu;
        s3++;
    }
    freopen("out.txt", "w", stdout);
    for (i = 0; i < s; i++) {
        count = 0;
        //求第5种疾病代号为e
        //P-D-P-D开始求解 对每种piRNA开始终点是e
        //s是结点个数 s1是序列个数 s2是疾病个数 s3是病序
        //P-D
        for (j = 0; j < s3; j++) {
            char ds;
            if (nodes[i] == gx[j].a) {
                ds = gx[j].c1;
                //D-P
                for (int k = 0; k < s3; k++) {
                    int ss;
                    if (ds == gx[k].c1 /*&& gx[k].a != nodes[i]*/) {
                        ss = gx[k].a;
                        //P-D
                        for (int l = 0; l < s3; l++) {
                            if (ss == gx[l].a && gx[l].c1 == 'e') {
                                count++;
                            }
                        }


                    }
                }
            }
        }
        cout << count << endl;

    }

    return 0;
}
```

第3种P-D-D（已手动验证）

```
#include<iostream>

using namespace std;
int node;
int count;
int nodes[5000];
struct xulie {
    int a, b;
} xulie[500749];
struct jibing {
    char c1, c2;
} dis[155];
struct guanxi {
    int a;
    char c1;
} gx[5002];

int main() {
    freopen("node.txt", "r", stdin);
    int s = 0, s1 = 0, s2 = 0, s3 = 0;
    while (cin >> node) {
        nodes[s++] = node;
    }
    cin.clear();
    freopen("xuliesim.txt", "r", stdin);
    int i, j;
    while (cin >> i >> j) {
        xulie[s1].a = i;
        xulie[s1].b = j;
        s1++;

    }
    cin.clear();
    char c1, c2;
    freopen("bing.txt", "r", stdin);
    while (cin >> c1 >> c2) {
        dis[s2].c1 = c1;
        dis[s2].c2 = c2;
        s2++;
    }
    cin.clear();
    freopen("bingxu.txt", "r", stdin);
    int shu;
    char fu;
    while (cin >> shu >> fu) {
        gx[s3].a = shu;
        gx[s3].c1 = fu;
        s3++;
    }
    freopen("out.txt", "w", stdout);
    for (i = 0; i < s; i++) {
        count = 0;
        //求第5种疾病代号为e
        //P-D-D开始求解 对每种piRNA开始终点是e
        //s是结点个数 s1是序列个数 s2是疾病个数 s3是病序
        //P-D-D
        //求D
        for (j = 0; j < s3; j++) {
            char d;
            if (nodes[i] == gx[j].a) {
                d = gx[j].c1;
                //求D-D
                for (int k = 0; k < s2; k++) {
                    if (d == dis[k].c1 && dis[k].c2 == 'e' || d == dis[k].c2 &&
dis[k].c1 == 'e') {
                        count++;
                    }
                }
            }
        }

        cout << count << endl;

    }

    return 0;
}
```

第4种P-P-P-D(手动验证)

```
#include<iostream>

using namespace std;
int node;
int count;
int nodes[5000];
struct xulie {
    int a, b;
} xulie[500749];
struct jibing {
    char c1, c2;
} dis[155];
struct guanxi {
    int a;
    char c1;
} gx[5002];

int main() {
    freopen("node.txt", "r", stdin);
    int s = 0, s1 = 0, s2 = 0, s3 = 0;
    while (cin >> node) {
        nodes[s++] = node;
    }
    cin.clear();
    freopen("xuliesim.txt", "r", stdin);
    int i, j;
    while (cin >> i >> j) {
        xulie[s1].a = i;
        xulie[s1].b = j;
        s1++;

    }
    cin.clear();
    char c1, c2;
    freopen("bing.txt", "r", stdin);
    while (cin >> c1 >> c2) {
        dis[s2].c1 = c1;
        dis[s2].c2 = c2;
        s2++;
    }
    cin.clear();
    freopen("bingxu.txt", "r", stdin);
    int shu;
    char fu;
    while (cin >> shu >> fu) {
        gx[s3].a = shu;
        gx[s3].c1 = fu;
        s3++;
    }
    freopen("out.txt", "w", stdout);
    for (i = 0; i < s; i++) {
        count = 0;
        //求第5种疾病代号为e
        //P-P-P-D开始求解 对每种piRNA开始终点是e
        //s是结点个数 s1是序列个数 s2是疾病个数 s3是病序
        //求P-P
        for (j = 0; j < s1; j++) {
            if (nodes[i] == xulie[j].a || nodes[i] == xulie[j].b) {
                int zd;
                if (nodes[i] == xulie[j].a) {
                    zd = xulie[j].b;
                } else
                    zd = xulie[j].a;
                //求 p-p
                for (int k = 0; k < s1; k++) {
                    if (zd == xulie[k].a || zd == xulie[k].b) {
                        int zd1;
                        if (zd == xulie[k].a) {
                            zd1 = xulie[j].b;
                        } else
                            zd1 = xulie[j].a;
                        //p-D
                        for (int l = 0; l < s3; l++) {
                            if (zd1 == gx[l].a && gx[l].c1 == 'e')
                                count++;
                        }
                    }
                }
            }
        }

        cout << count << endl;
    }

    return 0;
}
```

第5种P-D-D-D（已手动验证）

```
#include<iostream>

using namespace std;
int node;
int count;
int nodes[5000];
struct xulie {
    int a, b;
} xulie[500749];
struct jibing {
    char c1, c2;
} dis[155];
struct guanxi {
    int a;
    char c1;
} gx[5002];

int main() {
    freopen("node.txt", "r", stdin);
    int s = 0, s1 = 0, s2 = 0, s3 = 0;
    while (cin >> node) {
        nodes[s++] = node;
    }
    cin.clear();
    freopen("xuliesim.txt", "r", stdin);
    int i, j;
    while (cin >> i >> j) {
        xulie[s1].a = i;
        xulie[s1].b = j;
        s1++;

    }
    cin.clear();
    char c1, c2;
    freopen("bing.txt", "r", stdin);
    while (cin >> c1 >> c2) {
        dis[s2].c1 = c1;
        dis[s2].c2 = c2;
        s2++;
    }
    cin.clear();
    freopen("bingxu.txt", "r", stdin);
    int shu;
    char fu;
    while (cin >> shu >> fu) {
        gx[s3].a = shu;
        gx[s3].c1 = fu;
        s3++;
    }
    freopen("out.txt", "w", stdout);
    for (i = 0; i < s; i++) {
        count = 0;
        //求第5种疾病代号为e
        //P-D-D-D开始求解 对每种piRNA开始终点是e
        //s是结点个数 s1是序列个数 s2是疾病个数 s3是病序
        //P-D
        for (j = 0; j < s3; j++) {
            if (nodes[i] == gx[j].a) {
                //D-D
                char d;
                for (int k = 0; k < s2; k++) {
                    if (gx[i].c1 == dis[k].c1 || gx[i].c1 == dis[k].c2) {
                        if (gx[i].c1 == dis[k].c1) {
                            d = dis[k].c2;
                        } else
                            d = dis[k].c1;
                        //D-D
                        for (int l = 0; l < s2; l++) {
                            if (d == dis[l].c1 && dis[l].c2 == 'e' || d ==
dis[l].c2 && dis[l].c1 == 'e')
                                count++;
                        }
                    }
                }
            }
        }

        cout << count << endl;

    }

    return 0;
}
```

第6种 P-P-D-D（已手动验证）

```
#include<iostream>

using namespace std;
int node;
int count;
int nodes[5000];
struct xulie {
    int a, b;
} xulie[500749];
struct jibing {
    char c1, c2;
} dis[155];
struct guanxi {
    int a;
    char c1;
} gx[5002];

int main() {
    freopen("node.txt", "r", stdin);
    int s = 0, s1 = 0, s2 = 0, s3 = 0;
```

```cpp
    while (cin >> node) {
        nodes[s++] = node;
    }
    cin.clear();
    freopen("xuliesim.txt", "r", stdin);
    int i, j;
    while (cin >> i >> j) {
        xulie[s1].a = i;
        xulie[s1].b = j;
        s1++;

    }
    cin.clear();
    char c1, c2;
    freopen("bing.txt", "r", stdin);
    while (cin >> c1 >> c2) {
        dis[s2].c1 = c1;
        dis[s2].c2 = c2;
        s2++;
    }
    cin.clear();
    freopen("bingxu.txt", "r", stdin);
    int shu;
    char fu;
    while (cin >> shu >> fu) {
        gx[s3].a = shu;
        gx[s3].c1 = fu;
        s3++;
    }
    freopen("out.txt", "w", stdout);
    for (i = 0; i < s; i++) {
        count = 0;
        //求第5种疾病代号为e
        //P-P-D-D开始求解 对每种piRNA开始终点是e
        //s是结点个数 s1是序列个数 s2是疾病个数 s3是病序
        //P-P
        for (j = 0; j < s1; j++) {
            if (nodes[i] == xulie[j].a || nodes[i] == xulie[j].b) {
                int zd;
                if (nodes[i] == xulie[j].a)
                    zd = xulie[j].b;
                else
                    zd = xulie[j].a;
                //P-D
                for (int k = 0; k < s3; k++) {
                    if (zd == gx[k].a) {
                        //D-D
                        for (int l = 0; l < s2; l++) {
                            if (dis[l].c2 == gx[k].c1 && dis[l].c1 == 'e' ||
 dis[l].c1 == gx[k].c1 && dis[l].c2 == 'e')
                                count++;
                        }
                    }
                }
            }
        }
        cout << count << endl;
    }
    return 0;
}
```

分类代码

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import time
import numpy as np
from numpy import linalg as LA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix

from scipy import stats

import sys
from os import listdir
from os.path import isfile, join

# 判断是否输入了4个参数（个人注释）
# ./weightedRandomforest.py  ./input-15  weightedRandomforest  5



if (len(sys.argv) != 4):
    sys.exit("Usage: %s directory-for-input-files outdir randomSeed\n" %
(sys.argv[0]))


# 参数计算，定义calPRAUC函数，计算查准率，查全率，PR曲线（个人注释）

#precision 所有预测为真的里面包括多少个实际真的  预测了100个真的 里面只有30个真的预测为真的，
30%

#recall 所有实际真的里面将正类预测为真的      实际有100个真的  将其中99个真的预测为真的 99%

#求auc曲线下梯形的面积，也就是看该训练模型的区分能力

# 用PRAUC是因为查准率和查全率都集中分析正样本


 # prauc = calPRAUC(orderScores, y_p_test.shape[0], topN)  其中topN=500 ntp所有正样
本被测为负的概率
def calPRAUC(ranks, nTPs, topN):
    cumPRAUC = 0
    posRecalls = list()
    for i in range(topN):
        if (ranks[i] < nTPs):
            posRecalls.append(1)
        else:
            posRecalls.append(0)

    curSum = posRecalls[0]
    prevRecall = round(posRecalls[0] / nTPs, 4)
    prevPrec = round(posRecalls[0], 4)
    for i in range(1, topN):
        curSum += posRecalls[i]
        recall = round(curSum / nTPs, 4)
        # precision @k =TP/k
        prec = round(curSum / (i + 1), 4)
        cumPRAUC += ((recall - prevRecall) * (prevPrec + prec) / 2)
        prevRecall = recall
        prevPrec = prec

    cumPRAUC = round(cumPRAUC, 4)
    return cumPRAUC
```

```
# python ./weightedRandomforest.py ./input-15 weightedRandomforest 1（个人注释）
# 运行时传入四个参数，sys.argv[1]是对输入文件进行，rs是指运行次数（每种方法都使用不同的随机种子集运行10次。））
# outdir指输出结果的策略 nfolds指的是5倍交叉验证。
# 将数据分成5个箱子，并在箱子中保留相同百分比的正样本和未标记样本，每次旋转以其中一个作为测试集，而其他四个箱子作为训练集。我们对正未标记学习算法训练5次，在整个数据集中找到可靠的负样本，然后我们训练一个具有可靠的负样本和正标记样本的新分类器，用于预测未标记样本。
# topN = 500，为了评估一种方法的性能，只使用顶部的k个预测，动机是评估该方法在特定疾病的前k个预测中恢复正相关的能力。
# nTrees = 30 每个森林有30颗树，也就是n_estimators的个数
inpath = sys.argv[1] #输入路径
outdir = sys.argv[2] #输出目录
rs = int(sys.argv[3])  #运行随机种子编号

nfolds = 5
topN = 500
T = 30
nTrees = 30
# mFeats = [4, 5, 6, 7, 8, 9]
mFeats = [2, 3, 4, 5, 6, 7, 8, 10]
R = np.power(2.0, [5, 6, 7, 8, 9, 10, 11, 12])  #求2的n次方 求最优参数
# R = np.power(10.0, range(2, 7))

nC = len(mFeats)
nR = len(R)
dFeatures = [f for f in listdir(inpath) if isfile(join(inpath, f))]
for df in dFeatures:
    print("Processing %s" % (df))
    dId = df.split('_')[0]
    pf = "/".join((inpath, df))  # ./input-15/D001_features.csv

    outfile = ".".join((dId, "txt")) #D001.txt
    of = "/".join((outdir, outfile))    #weightRandomforest/D001.txt
    #    featImp = ".".join((dId, "impFeat")) #feature importance
    #    fif = "/".join((outdir, featImp))
    # 对输入的csv文件进行操作（个人注释）
    eRecalls = np.zeros(nfolds) #返回5个全0的数
    ePrecisions = np.zeros(nfolds)    #返回5个全0的数
    ePRAUCs = np.zeros(nfolds) #返回5个全0的数
    d = np.loadtxt(pf, delimiter=',') #对pf是路径  csv文件进行输出
    #注意：X[:, m:n]即取矩阵X的所有行中的的第m到n-1列数据，含左不含右
    tt=6  #不去 是6       去1是5       去2是4        去3是3        去4是2
    xss=7
    p = d[d[:, tt] == 1, :]  #所有行第6列元素等于1
    u = d[d[:, tt] == 0, :]  #所有行第6列元素等于0的行
    X_p = p[:, 0:tt]      # X_p正样本0-6列
    y_p = p[:, tt]         # y_p正样本第7列
    X_u = u[:, 0:tt]       #未标记样本0-6列
    y_u = u[:, tt]           #未标记样本第7列
    xx = d[:,xss]    #专门第7列
    y_u = np.ones(y_u.shape[0]) * -1  #将y_u中未标记样本的值0都赋值为-1
    # nfolds to evaluate the performance
    # ikf是第几倍交叉验证
    ikf = 0
    #shuffle,这其实在python里面也是一个函数，就是打乱顺序的意思。
    # 那么它的作用是什么呢？比如说我们现在是将索引从0-19的20个索引利用K折验证方法，也就是说要
将数据分成两块。
    # 如果按照default shuffle = False，
    # 那么分法就是按照顺序分，也就是第一个的验证集是0-9，第二个是10-19。如果shuffle = True，
那么则表示将打乱顺序后再进行分配。比如我对n_splits = 2 时候根据shuffle 的boolean值不同进行
了测试
    #随机种子是保持不同的结果
    kf = KFold(n_splits=nfolds, shuffle=False, random_state=rs)
    X_p_splits = list(kf.split(X_p)) #将正样本分为训练集和测试集
    X_u_splits = list(kf.split(X_u)) #将未标记样本分为训练集和测试集



    #自己----------------------------------------------------------------------------
----------修改部分
    allpositive = p[:, xss]  # 序号
    allunknown = u[:, xss]  # 序号
    # print(allpositive[50])
    sum_positive = 0
    sum_unknown = 0







    for ikf in range(nfolds):
        p_train_index, p_test_index = X_p_splits[ikf]  #测试集是0
        u_train_index, u_test_index = X_u_splits[ikf]
        X_p_train = X_p[p_train_index]
        y_p_train = y_p[p_train_index]
        X_p_test = X_p[p_test_index]
        y_p_test = y_p[p_test_index]

        X_u_train = X_u[u_train_index]
        y_u_train = y_u[u_train_index]
        X_u_test = X_u[u_test_index]
        y_u_test = y_u[u_test_index]
        #print("Train:", X_p_train.shape, "test:", X_p_test.shape)

        start_time = time.time()
        cvMeans = np.zeros(nC * nR)
        cvStds = np.zeros(nC * nR)
        ithPair = 0
        # nested nfolds to select optimal parameters
        kf2 = KFold(n_splits=nfolds, shuffle=False, random_state=rs)
        for mf in mFeats:
            for r in R:
                recalls = np.zeros(nfolds)  # recall rate per each c-r pair
                X_p_cv_splits = list(kf2.split(X_p_train))
                X_u_cv_splits = list(kf2.split(X_u_train))
                for ikf2 in range(nfolds):
                    p_train_cv_index, p_val_cv_index = X_p_cv_splits[ikf2]
                    u_train_cv_index, u_val_cv_index = X_u_cv_splits[ikf2]
                    X_p_cv_train = X_p_train[p_train_cv_index]
                    y_p_cv_train = y_p_train[p_train_cv_index]
                    X_p_cv_val = X_p_train[p_val_cv_index]
                    y_p_cv_val = y_p_train[p_val_cv_index]

                    X_u_cv_train = X_u_train[u_train_cv_index]
                    y_u_cv_train = y_u_train[u_train_cv_index]
                    X_u_cv_val = X_u_train[u_val_cv_index]
                    y_u_cv_val = y_u_train[u_val_cv_index]

                    # mix validation + unlabel for transductive learning to see
how it perform on validation set
                    X_pu_cv_train = np.concatenate((X_p_cv_train, X_u_cv_train),
axis=0)
                    y_pu_cv_train = np.concatenate((y_p_cv_train, y_u_cv_train),
axis=0)
                    X_pu_cv_val = np.concatenate((X_p_cv_val, X_u_cv_val),
axis=0)
                    y_pu_cv_val = np.concatenate((y_p_cv_val, y_u_cv_val),
axis=0)
                    scaler = StandardScaler().fit(X_pu_cv_train)
                    X_pu_cv_train_transformed = scaler.transform(X_pu_cv_train)
                    #                     pca = PCA(0.99, svd_solver="full",
random_state = 0)
                    #                     pca.fit(X_pu_cv_train_transformed)
                    #                     X_pu_cv_train_transformed =
pca.transform(X_pu_cv_train_transformed)
                    X_pu_cv_val_transformed = scaler.transform(X_pu_cv_val)
                    #                     X_pu_cv_val_transformed =
pca.transform(X_pu_cv_val_transformed)
                    # oob_score = False 不使用袋外样品进行估算
                    clf = RandomForestClassifier(n_estimators=nTrees,
max_depth=mf, oob_score=False,
                                                 class_weight={-1: 1, 1: r},
random_state=1)
                    # clf = RandomForestClassifier(n_estimators = nTrees,
max_features = mf, oob_score = False, class_weight = 'balanced', random_state =
1)
                    clf.fit(X_pu_cv_train_transformed, y_pu_cv_train)
                    #分数 取第2列，即分类为1的序列
                    scores = clf.predict_proba(X_pu_cv_val_transformed)[:, 1]
                    # -scores从小到大排列，即越靠前的预测越准确
                    orderScores = np.argsort(-scores)
                    topNIndex = orderScores[:topN]
                    # print("avgScores:", avgScores[topNIndex])
                    truePosIndex = np.array(range(y_p_cv_val.shape[0]))
                    truePosRecall = np.intersect1d(topNIndex, truePosIndex,
assume_unique=True)
                    recall = truePosRecall.shape[0] / truePosIndex.shape[0]
                    #                     recall = calPRAUC(orderScores,
y_p_cv_val.shape[0], topN)
                    recalls[ikf2] = recall
                    # scores = clf.predict(X_pu_cv_val_transformed)
                    # nPos = np.sum(scores == 1)
                    # if nPos == 0:
                    #    nPos = 1
                    # posRate = nPos / y_pu_cv_val.shape[0]
                    # recall = recall_score(y_pu_cv_val, scores)
                    # recalls[ikf2] = recall * recall / posRate
                    # print("For mf: %d, fold:%d, recall:%f, F' measure: %f " %
(mf, ikf2, recall, recalls[ikf2]))
                    # print(confusion_matrix(y_pu_cv_val, scores))
                avgRecall = np.mean(recalls)
                cvMeans[ithPair] = avgRecall
                stdRecall = np.std(recalls)
                cvStds[ithPair] = stdRecall
                #                print("For mfeatures: %d, class_weight ratio:
%f, rank of top %d: average recall: %.2f%%, std of recall: %.2f" %(mf, r, topN,
avgRecall*100, stdRecall ))
                #                print("For each fold:", recalls)
                ithPair += 1
        elapsed_time = time.time() - start_time
        cvMaxMeanIndex = np.argmax(cvMeans)
        optimalM = mFeats[cvMaxMeanIndex // nR]
        optimalR = R[cvMaxMeanIndex % nR]
        #       print("cv-MaxMean:", cvMeans[cvMaxMeanIndex], "cv-MaxMean_std:",
cvStds[cvMaxMeanIndex], "cvMaxMeanIndex:", cvMaxMeanIndex)
        print("disease:", dId, ", randomSeed:", rs, ", ithFold:", ikf, ",
optimalM:", optimalM, ", optimalR:", optimalR,
              ", cv-MaxMean:", cvMeans[cvMaxMeanIndex])
        #        print("cross-validation time elapsed: %.2f" %(elapsed_time) )
        # After parameter selection, we evaluate on the test set with the
optimal parameters
        X_test = np.concatenate((X_p_test, X_u_test), axis=0)    #拼接函数 axis等于
0为纵向拼接
        y_test = np.concatenate((y_p_test, y_u_test), axis=0)  #y是指类别
        X_train = np.concatenate((X_p_train, X_u_train), axis=0)
        y_train = np.concatenate((y_p_train, y_u_train), axis=0)
        scaler = StandardScaler().fit(X_train)
        X_train_transformed = scaler.transform(X_train)
        #        pca = PCA(0.99, svd_solver="full", random_state = 0)
        #        pca.fit(X_train_transformed)
        #        X_train_transformed = pca.transform(X_train_transformed)
        X_test_transformed = scaler.transform(X_test)
        #        X_test_transformed = pca.transform(X_test_transformed)
        clf = RandomForestClassifier(n_estimators=nTrees, max_depth=optimalM,
oob_score=False,
                                    class_weight={-1: 1, 1: optimalR},
random_state=1) #-1占比1 然后optimalR占比32
        # clf = RandomForestClassifier(n_estimators = nTrees, max_features =
optimalM, oob_score = False, class_weight = 'balanced', random_state = 1)
        clf.fit(X_train_transformed, y_train) #训练数据
        # scores = clf.predict(X_test_transformed)
        #输出第2列所有为真的那列
        #print(X_test_transformed)
        scores = clf.predict_proba(X_test_transformed)[:, 1]
       # print(scores)
        #        scoreList = [str(item) for item in scores]
        #        scoreStr = ','.join(scoreList)
        # recall = recall_score(y_test, scores)
        orderScores = np.argsort(-scores)

        # ------------------------------------------------------------------------
---------------------------------------
        #print(orderScores)
        ouput_scores=[]
        for i in orderScores:  # 24 22
            if i < len(p_test_index):
                str0 = str(allpositive[i + sum_positive])
                str1 = str0 + "\t" + str(round(scores[i], 6))


            else:
                str0 = str(allunknown[i - len(p_test_index) + sum_unknown])
                str1 = str0 + "\t" + str(round(scores[i], 6))
            ouput_scores.append(str1)
        #print(ouput_scores)
        orderStr = ','.join(ouput_scores)
        sum_positive += len(p_test_index)
        sum_unknown += len(u_test_index)

        #-------------------------------------------------------------------------
--------------------------------------



        #orderList = [str([xx[item+ikf*870]]) for item in orderScores]
        # orderStr = ','.join(orderList)

        topNIndex = orderScores[:topN]
        #print("avgScores:", avgScores[topNIndex])
        #42等于测试集中行数
        truePosIndex = np.array(range(y_p_test.shape[0]))  #测试集的行数
        #topNIndex和真正的交集筛掉truePosINdex
        # TP是所有预测为正的-假的预测为正的   故truePosIndex为假的预测为正的
        # a = np.array([8, 2, 3, 4, 2, 4, 1])
        # b = np.array([7, 9, 5, 6, 3])
        # c = np.setdiff1d(a, b, True)
        # print(c)  # [8 2 4 2 4 1]
        #即truePosIndex为假的预测为正的
        # topNIndex为全部预测为正的
        truePosRecall = np.intersect1d(topNIndex, truePosIndex,
assume_unique=True)
        # 42/42
        # TP/(TP+FN)  即truePosRecall.shape[0] 为TP
        recall = truePosRecall.shape[0] / truePosIndex.shape[0]
        #truePosRecall是真实的五分之一
        precision = truePosRecall.shape[0] / topN #真实的42/500
        prauc = calPRAUC(orderScores, y_p_test.shape[0], topN)
        eRecalls[ikf] = recall
        ePrecisions[ikf] = precision
        ePRAUCs[ikf] = prauc
        ssddasdsf="\n".join(ouput_scores)
        print("dId: %s, randomState: %d, %dth-fold, recall: %.2f%%, precision: 
%.2f%%, prauc: %.4f" % (
        dId, rs, ikf, recall * 100, precision * 100, prauc))
        with open(of, "a") as output:
            output.write("%s-RandomState%d-%dth fold, number of true 
positive:%d\n" % (dId, rs, ikf, y_p_test.shape[0]))
            output.write("%s\n" % (ssddasdsf))
            output.write("END\n")
    mRecall = np.mean(eRecalls)
    stdRecall = np.std(eRecalls)
    mPrec = np.mean(ePrecisions)
    stdPrec = np.std(ePrecisions)
    mPRAUC = np.mean(ePRAUCs)
    stdPRAUC = np.std(ePRAUCs)
    recallList = [str(item) for item in eRecalls]
    precList = [str(item) for item in ePrecisions]
    praucList = [str(item) for item in ePRAUCs]
    recallStr = ','.join(recallList)
    precStr = ','.join(precList)
    praucStr = ','.join(praucList)
    with open(of, "a") as output:
        output.write("%s-RandomState%d, mean+-std recall:%.4f,%.4f\n" % (dId, 
rs, mRecall, stdRecall))
        output.write("%s-RandomState%d, mean+-std precision:%.4f,%.4f\n" % (dId, 
rs, mPrec, stdPrec))
        output.write("%s-RandomState%d, mean+-std prauc:%.4f,%.4f\n" % (dId, rs, 
mPRAUC, stdPRAUC))
        output.write("%s-RandomState%d, 5-fold cv recall:%s\n" % (dId, rs, 
recallStr))
        output.write("%s-RandomState%d, 5-fold cv precision:%s\n" % (dId, rs, 
precStr))
        output.write("%s-RandomState%d, 5-fold cv prauc:%s\n" % (dId, rs, 
praucStr))
        output.write("END\n")
    print(
        "summary of %s, randomSeed: %d, top %d, mean/std of prauc, mean/std of 
recall, mean/std of precision: %f,%f,%f,%f,%f,%f" % (
        dId, rs, topN, mPRAUC, stdPRAUC, mRecall, stdRecall, mPrec, stdPrec))
    print(eRecalls)
    print(ePrecisions)
    print(ePRAUCs)
    print("END")
```

求相似度算法

```
#include <bits/stdc++.h>

using namespace std;
map<char, int> valueKey;
```

```cpp
void keyInit() {
  valueKey['A'] = 0;
  valueKey['C'] = 1;
  valueKey['G'] = 2;
  valueKey['T'] = 3;
  valueKey['-'] = 4;
}

int mapping[5][5] = {5,  -1, -2, -1, -3, -1, 5,  -3, -2, -4, -2, -3, 5,
                     -2, -2, -1, -2, -2, 5,  -1, -3, -4, -2, -1, 0};
int dp[101][101];
int dpcnt[101][101];

int max(int x, int y, int z) {
  int temp = x > y ? x : y;
  return temp > z ? temp : z;
}

int min(int x, int y) { return x < y ? x : y; }

class DNA {
 public:
  string name;
  string s;

  DNA(string name, string s) : name(name), s(s) {}
};

double func(DNA a, DNA b) {
  int l1 = a.s.size();
  int l2 = b.s.size();
  string s1 = a.s;
  string s2 = b.s;
  dp[0][0] = 0;
  dpcnt[0][0] = 0;
  for (int i = 1; i <= l1; i++) {
    dp[i][0] = dp[i - 1][0] + mapping[valueKey[s1[i]]][valueKey['-']];
    dpcnt[i][0] = dpcnt[i - 1][0] + 1;
  }
  for (int j = 1; j <= l2; j++) {
    dp[0][j] = dp[0][j - 1] + mapping[valueKey['-']][valueKey[s2[j]]];
    dpcnt[0][j] = dpcnt[0][j - 1] + 1;
  }
  for (int i = 1; i <= l1; i++) {
    for (int j = 1; j <= l2; j++) {
      dp[i][j] =
          max(dp[i][j - 1] + mapping[valueKey['-']][valueKey[s2[j]]],
              dp[i - 1][j] + mapping[valueKey[s1[i]]][valueKey['-']],
              dp[i - 1][j - 1] + mapping[valueKey[s1[i]]][valueKey[s2[j]]]);
      if (dp[i][j] ==
          dp[i - 1][j - 1] + mapping[valueKey[s1[i]]][valueKey[s2[j]]])
        dpcnt[i][j] = dpcnt[i - 1][j - 1] + 1;
      else if (dp[i - 1][j] + mapping[valueKey[s1[i]]][valueKey['-']] ==
               dp[i][j - 1] + mapping[valueKey['-']][valueKey[s2[j]]]) {
        dpcnt[i][j] = min(dpcnt[i - 1][j], dpcnt[i][j - 1]) + 1;
      } else if (dp[i][j] ==
                 dp[i - 1][j] + mapping[valueKey[s1[i]]][valueKey['-']])
        dpcnt[i][j] = dpcnt[i - 1][j] + 1;
      else
        dpcnt[i][j] = dpcnt[i][j - 1] + 1;
    }
  }
  return 20.0 * dp[l1][l2] / dpcnt[l1][l2];
}
int main() {
  keyInit();
  freopen("a.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  string name, s;
  vector<DNA> a;
  while (cin >> name >> s) {
    DNA t = DNA(name, s);
    a.push_back(t);
  }
  for (int i = 0; i < a.size() - 1; i++) {
    for (int j = i + 1; j < a.size(); j++) {
      double t = func(a[i], a[j]);
        if(t>30){
                cout << a[i].name << "\t\t" << a[j].name << "\t\t" << fixed
           << setprecision(2) << t << "%" << endl;
        }

      //printf("%s\t\t%s\t\t%.2f%%\n", a[i].name.c_str(), a[j].name.c_str(), t);
    }
  }
  return 0;
}
```