A method for the definition of immunological non-response to antiretroviral therapy based on review analysis and supervised classification model

——Supplement (Technical details of the whole modeling process)

1. Visual map modeling method of features related to the definition of INR

```
from py2neo import Graph,Node,Relationship
#V 5.0 need this pattern
test_graph = Graph("http://localhost:7474", auth=("neo4j", "123456"))

test_graph.delete_all()
# print(test_graph.)
# Create Node
# INR
inr=Node("INR",lable='result',name="INR")
# reason
c_ab=Node("CD4",lable='reason',name="CD4+ T-cell count absolute value")
c_ad=Node("CD4",lable='reason',name="CD4+ T-cell count change value")
c_ap=Node("CD4",lable='reason',name="CD4+ T-cell count growth rate")
c_48=Node("CD4",lable='reason',name="CD4/CD8 ratio")
art_time=Node("time",lable='reason',name="ART time ")
vs_time=Node("time",lable='reason',name="VS time ")
#add item
test_graph.create(inr)
test_graph.create(c_ab)
test_graph.create(c_ad)
test_graph.create(c_ap)
test_graph.create(c_48)
test_graph.create(art_time)
test_graph.create(vs_time)
#create relationship
relation1=Relationship(c_ab,"define",inr)
relation2=Relationship(c_ad,"define",inr)
relation3=Relationship(c_ap,"define",inr)
relation4=Relationship(c_48,"define",inr)
relation5=Relationship(art_time,"define",inr)
relation6=Relationship(vs_time,"define",inr)

test_graph.create(relation1)
test_graph.create(relation2)
test_graph.create(relation3)
test_graph.create(relation4)
test_graph.create(relation5)
```

```
test_graph.create(relation6)
```

2. Supervised learning modeling

2.1 Public data access

```
import pymssql
class    Dataoperator:
    def runsql_search(self):
        conn = pymssql.connect(server='127.0.0.1', port=1433, user='sa', password='alex0414',
                               database='AI_AIDS_Data')
        cursor_1 = conn.cursor()
        cursor_1.execute(self)

        return (cursor_1.fetchall())
    def runsql_insert(self,insertcontent):
        conn = pymssql.connect(server='127.0.0.1', port=1433, user='sa', password='alex0414',
                               database='AI_AIDS_Data')
        cursor=conn.cursor()
        cursor.executemany(self,insertcontent)
        conn.commit()
        conn.close()

def loadFeaturename_fromdatatable(sql):
    dt = Dataoperator.runsql_search(sql)
    if(dt==None or len(dt)==0):
        return None
    else:
        fr=dt
    return np.array(fr)
sql0="select * from inr_supvised where inr!=''"
df_1=loadFeaturename_fromdatatable(sql0)
df_1=np.array(df_1)
df_1=df_1.tolist()
df=pd.read_excel('inr_data.xlsx')
df=df.values.tolist()
df=np.array(df)
df=df.tolist()
df_total=np.append(df,df_1,axis=0)
df_total=np.array(df_total)
```

2.2 KNN

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.svm import SVC,LinearSVC
from sklearn.neighbors import KNeighborsClassifier
```

```
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

#preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_total)
df_total=scaler.transform(df_total)

# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
for algorithm in ['brute','kd_tree','ball_tree','auto']:
    for leaf_size in [10,20,30,40,50]:
        if(algorithm!='ball_tree' and algorithm!='kd_tree'):
            continue
        for n_neighbors in [2,3,4,5,6,7,8,9]:
            for metric in ['euclidean','manhattan','chebyshev','minkowski']:
                for weights in ['uniform','distance']:
                    shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                    knn = KNeighborsClassifier(algorithm=algorithm, n_neighbors=n_neighbors,
metric=metric, weights=weights, leaf_size=leaf_size)
                    score = cross_val_score(knn, df_total[:, :2], df_total[:, 2], cv=shuf)
                    my_paras = {'algorithm': algorithm, 'leaf_size': leaf_size, 'n_neighbors':
n_neighbors, 'metric': metric, 'weights': weights}
                    if score.mean() > best_score:
                        best_score = score.mean()
                        best_parameters = my_paras
                        print("the score of {} from best para
{}".format(best_score,best_parameters))

print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'algorithm': 'ball_tree', 'leaf_size': 10, 'n_neighbors': 2, 'metric': 'chebyshev',
'weights': 'distance'}
best_score is 0.9854907975460122
'''
```

2.3 Lasso

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.linear_model import Lasso
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
for alpha in [0.001, 0.01, 0.1, 1, 10, 100]:
    for selection in ['random','cyclic']:
        for max_iter in [10, 100, 1000, 10000]:
            for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
                try:
                    my_paras = {'alpha': alpha, 'selection': selection, 'max_iter': max_iter, 'tol': tol}
                    shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                    las = Lasso(alpha=alpha, selection=selection, max_iter=max_iter,
tol=tol,normalize=True,random_state=42)
                    score = cross_val_score(las, df_total[:, :2], df_total[:, 2], cv=shuf)
                    if score.mean() > best_score:
                        best_score = score.mean()
                        best_parameters = my_paras
                        print("the score of {} from best para {}".format(best_score,
best_parameters))
                except:
                    print('error')
                    print(my_paras)


print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'alpha': 0.001, 'selection': 'cyclic', 'max_iter': 10000, 'tol': 0.0001}
best_score is 0.5948517810797678
'''
```

2.4 MLP

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.neural_network import MLPClassifier
```

```
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

#preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_total)
df_total=scaler.transform(df_total)

# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
'''activation="relu", *,
                        solver='adam', alpha=0.0001,
                        batch_size='auto', learning_rate="constant",
                        learning_rate_init=0.001, power_t=0.5, max_iter=200,
                        shuffle=True, random_state=None, tol=1e-4,
                        verbose=False, warm_start=False, momentum=0.9,
                        nesterovs_momentum=True, early_stopping=False,
                        validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
                        epsilon=1e-8, n_iter_no_change=10, max_fun=15000'''
for activation    in ['identity','logistic', 'tanh', 'relu']:
      for solver in ['lbfgs','sgd','adam']:
            for alpha    in [1,0.1,0.01, 0.001, 0.0001, 0.00001]:
                  for learning_rate in ['constant','invscaling','adaptive']:
                        for max_iter in [10, 100, 1000]:
                              for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
                                    for hidden_layer_sizes    in [[10,10],[20,20],[30,30],[40,40],[50,50]]:
                                          shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                                          if (solver != 'sgd'):
                                                mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
activation=activation,
                                                                    solver=solver, alpha=alpha,
max_iter=max_iter, tol=tol,learning_rate='constant',random_state=42)
                                          else:
                                                mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
activation=activation,
                                                                    solver=solver, alpha=alpha,
max_iter=max_iter, tol=tol,learning_rate=learning_rate,random_state=42)

                                          score = cross_val_score(mlp, df_total[:, :2], df_total[:, 2], cv=shuf)
```

```
                                    my_paras = {'hidden_layer_sizes': hidden_layer_sizes, 'activation':
activation, 'solver': solver, 'alpha': alpha, 'learning_rate': learning_rate, 'max_iter': max_iter, 'tol': tol}
                                    if score.mean() > best_score:
                                        best_score = score.mean()
                                        best_parameters = my_paras
                                        print("the score of {} from best para {}".format(best_score,
best_parameters))
                                    else:
                                        print("the score of {} from No GOOD para
{}".format(score.mean(), my_paras))

print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'hidden_layer_sizes': [20, 20], 'activation': 'identity', 'solver': 'sgd', 'alpha': 1,
'learning_rate': 'adaptive', 'max_iter': 10000, 'tol': 1e-05}
best_score is 0.9674846625766871
'''
```

2.5 SVM

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.svm import SVC,LinearSVC
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

#preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_total)
df_total=scaler.transform(df_total)

# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
for kernel in ['linear','poly','rbf','sigmoid','precomputed']:
    for gamma in [0.01,0.1,1,10,100]:
        if(kernel=='linear' or kernel=='precomputed'):
```

```
                continue
            for C in [100, 10, 1, 0.1, 0.01,0.01]:
                for max_iter in [10, 100, 1000, 10000]:
                    for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
                        shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                        svc = SVC(kernel=kernel, gamma=gamma, C=C, max_iter=max_iter,
tol=tol,random_state=42)
                        score = cross_val_score(svc, df_total[:, :2], df_total[:, 2], cv=shuf)
                        my_paras = {'kernel': kernel, 'gamma': gamma, 'C': C, 'max_iter': max_iter,
'tol': tol}
                        if score.mean() > best_score:
                            best_score = score.mean()
                            best_parameters = my_paras
                            print("the score of {} from best para
{}".format(best_score,best_parameters))

print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:

best_parameters is {'kernel': 'rbf', 'gamma': 10, 'C': 100, 'max_iter': 10000, 'tol': 0.0001}
best_score is 0.991073619631902
'''
```

2.6 DT

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

#preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_total)
df_total=scaler.transform(df_total)

# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
```

```
for criterion in ['gini','entropy']:
    for splitter in ['best','random']:
        for max_depth in range(5, 100, 5):
            for min_samples_leaf in range(5, 100, 5):
                shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                dt = DecisionTreeClassifier(criterion=criterion, splitter=splitter,max_depth=max_depth,
min_samples_leaf=min_samples_leaf,random_state=42)
                score = cross_val_score(dt, df_total[:, :2], df_total[:, 2], cv=shuf)
                my_paras = {'criterion': criterion, 'splitter': splitter, 'max_depth': max_depth,
'min_samples_leaf': min_samples_leaf}
                if score.mean() > best_score:
                    best_score = score.mean()
                    best_parameters = my_paras
                    print("the score of {} from best para {}".format(best_score, best_parameters))


print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'criterion': 'entropy', 'splitter': 'best', 'max_depth': 5, 'min_samples_leaf': 5}
best_score is 0.9460736196319017
'''
```

2.7 Ridge

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.linear_model import Ridge
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

# #preprocessing
# from sklearn.preprocessing import MinMaxScaler
# scaler=MinMaxScaler()
# scaler.fit(df_total)
# df_total=scaler.transform(df_total)

# In Ridge, normalize=true indicates standardization of features
#    Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
```

```
for alpha in [0.001, 0.01, 0.1, 1, 10, 100]:
    for solver in ['svd','cholesky','sparse_cg','lsqr','sag']:
        for max_iter in [10, 100, 1000, 10000]:
            for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
                try:
                    my_paras = {'alpha': alpha, 'solver': solver, 'max_iter': max_iter, 'tol': tol}
                    shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                    rdg = Ridge(alpha=alpha, solver=solver, max_iter=max_iter,
tol=tol,normalize=True,random_state=42)
                    score = cross_val_score(rdg, df_total[:, :2], df_total[:, 2], cv=shuf)
                    if score.mean() > best_score:
                        best_score = score.mean()
                        best_parameters = my_paras
                        print("the score of {} from best para {}".format(best_score,
best_parameters))
                except:
                    print('error')
                    print(my_paras)


print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'alpha': 0.001, 'solver': 'cholesky', 'max_iter': 1000, 'tol': 1e-06}
best_score is 0.598587182698413
'''
```

2.8 GB Regression Tree 名称确定---对照数确定

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

#preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_total)
df_total=scaler.transform(df_total)
```

```
# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
for learning_rate in [0.001,0.005,0.01,0.05,0.1]:
    for n_estimators in [20,30,40,50,60,70,80,90,100]:
        for max_depth in range(5, 15, 1):
            for min_samples_split in range(200, 1000, 200):
                for min_samples_leaf in [30,40,50,60,70]:
                    shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                    gbc = GradientBoostingClassifier(learning_rate=learning_rate,
n_estimators=n_estimators,
                                                     max_depth=max_depth,
min_samples_split=min_samples_split,

min_samples_leaf=min_samples_leaf,random_state=42)
                    score = cross_val_score(gbc, df_total[:, :2], df_total[:, 2], cv=shuf)
                    my_paras = {'learning_rate': learning_rate, 'n_estimators': n_estimators,
'max_depth': max_depth,
                                'min_samples_split': min_samples_split, 'min_samples_leaf':
min_samples_leaf                                                    }
                    if score.mean() > best_score:
                        best_score = score.mean()
                        best_parameters = my_paras
                        print("the score of {} from best para {}".format(best_score,
best_parameters))



print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'learning_rate': 0.001, 'n_estimators': 90, 'max_depth': 5, 'min_samples_split':
800, 'min_samples_leaf': 60}
best_score is 0.7118098159509203
'''
```

2.9 LR

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.svm import SVC,LinearSVC
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
```

```
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')


#preprocessing
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaler.fit(df_total)
df_total=scaler.transform(df_total)
# Grid search with cross validation
best_score=np.float64(0)
best_parameters={}
for solver in ['newton-cg','sag','lbfgs','liblinear']:
    for penalty in ['l1', 'l2']:
        if(solver!='liblinear' and penalty=='l1'):
            continue
        for C in [100, 10, 1, 0.1, 0.01]:
            for max_iter in [10, 100, 1000, 10000]:
                for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
                    shuf = ShuffleSplit(test_size=0.5, train_size=0.3, n_splits=100)
                    lr = LogisticRegression(solver=solver, penalty=penalty, C=C, max_iter=max_iter,
tol=tol,random_state=42)
                    score = cross_val_score(lr, df_total[:, :2], df_total[:, 2], cv=shuf)
                    my_paras = {'solver': solver, 'penalty': penalty, 'C': C, 'max_iter': max_iter,
'tol': tol}
                    if score.mean() > best_score:
                        best_score = score.mean()
                        best_parameters = my_paras
                        print("the score of {} from best para
{}".format(best_score,best_parameters))

print('best_parameters is {}'.format(best_parameters))
print('best_score is {}'.format(best_score))

'''
result:
best_parameters is {'solver': 'liblinear', 'penalty': 'l1', 'C': 10, 'max_iter': 1000, 'tol': 0.01}
best_score is 0.9669018404907976

'''
```

3. Semi-supervised learning model

3.1 Model assumptions

①The open source data in reference [1] was real data, used for supervised training

②The electronic medical record database is used for unsupervised training

3.2 Modeling process

3.2.1 semi-kmeans

```
import pymssql
import numpy as np
import pandas as pd
np.set_printoptions(threshold=np.inf)

class    Dataoperator:
    def runsql_search(self):
        conn = pymssql.connect(server='127.0.0.1', port=1433, user='sa', password='alex0414',
                               database='AI_AIDS_Data')
        cursor_1 = conn.cursor()
        cursor_1.execute(self)

        return (cursor_1.fetchall())
    def runsql_insert(self,insertcontent):
        conn = pymssql.connect(server='127.0.0.1', port=1433, user='sa', password='alex0414',
                               database='AI_AIDS_Data')
        cursor=conn.cursor()
        cursor.executemany(self,insertcontent)
        conn.commit()
        conn.close()

def loadFeaturename_fromdatatable(sql):
    dt = Dataoperator.runsql_search(sql)
    if(dt==None or len(dt)==0):
        return None
    else:
        fr=dt
    return np.array(fr)
sql0="select * from inr_supvised where inr!=''"
U=loadFeaturename_fromdatatable(sql0)
U=np.array(U)
L=pd.read_excel('inr_data.xlsx')
L=L.values.tolist()
L=np.array(L)



def distEclud(vecA, vecB):
    '''
    Input: vectors A and B
```

```
    Output: Euclidean distance between A and B
    '''
    return np.sqrt(sum(np.power(vecA - vecB, 2)))


def newCent(L):
    '''
    Input: Labeled data set L
    Output: Determine the initial cluster center according to L
    '''
    centroids = []
    label_list = np.unique(L[:, -1])
    for i in label_list:
        L_i = L[(L[:, -1]) == i]
        cent_i = np.mean(L_i, 0)
        centroids.append(cent_i[:-1])
    return np.array(centroids)


def semi_kMeans(L, U, distMeas=distEclud, initial_centriod=newCent):
    '''
    Input: labeled data set L (the last column is the category label), unlabeled data set U (no category
label)
    Output: clustering results
    '''
    dataSet = np.vstack((L[:, :-1], U))    # Combine L and U
    label_list = np.unique(L[:, -1])
    k = len(label_list)    # Number of categories in L
    m = np.shape(dataSet)[0]

    clusterAssment = np.zeros(m)    # Initial sample allocation
    centroids = initial_centriod(L)    # Determine the initial cluster center
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        for i in range(m):    # Assign each sample to the nearest cluster center
            minDist = np.inf;
            minIndex = -1
            for j in range(k):
                distJI = distMeas(centroids[j, :], dataSet[i, :])
                if distJI < minDist:
                    minDist = distJI;
                    minIndex = j
            if clusterAssment[i] != minIndex: clusterChanged = True
```

```
                    clusterAssment[i] = minIndex
        return clusterAssment



clusterResult = semi_kMeans(L, U)
import matplotlib.pyplot as plt
x1=[]
x2=[]
y1=[]
y2=[]
for i in range(len(L)):
        if(L[i,2]==0):
                x1.append(L[i,1])
                y1.append(L[i,0])
        else:
                x2.append(L[i,1])
                y2.append(L[i,0])

x3=[]
x4=[]
y3=[]
y4=[]

for i in range(len(U)):
        if(clusterResult[i]==0):
                x3.append(U[i,1])
                y3.append(U[i,0])
        else:
                x4.append(U[i,1])
                y4.append(U[i,0])

colors1 = 'black'
colors2 = 'blue'
colors3 = 'red'
colors4 = 'green'
area = np.pi * 4**2
plt.scatter(x1, y1, s=area, c=colors1, alpha=0.4, label='INR-Labeled(Black)')
plt.scatter(x2, y2, s=area, c=colors2, alpha=0.4, label='IR-Labeled(Blue)')
plt.scatter(x3, y3, s=area, c=colors3, alpha=0.4, label='INR-Unlabeled(Red)')
plt.scatter(x4, y4, s=area, c=colors4, alpha=0.4, label='IR-Unlabeled(Green)')

plt.legend()
plt.show()
```

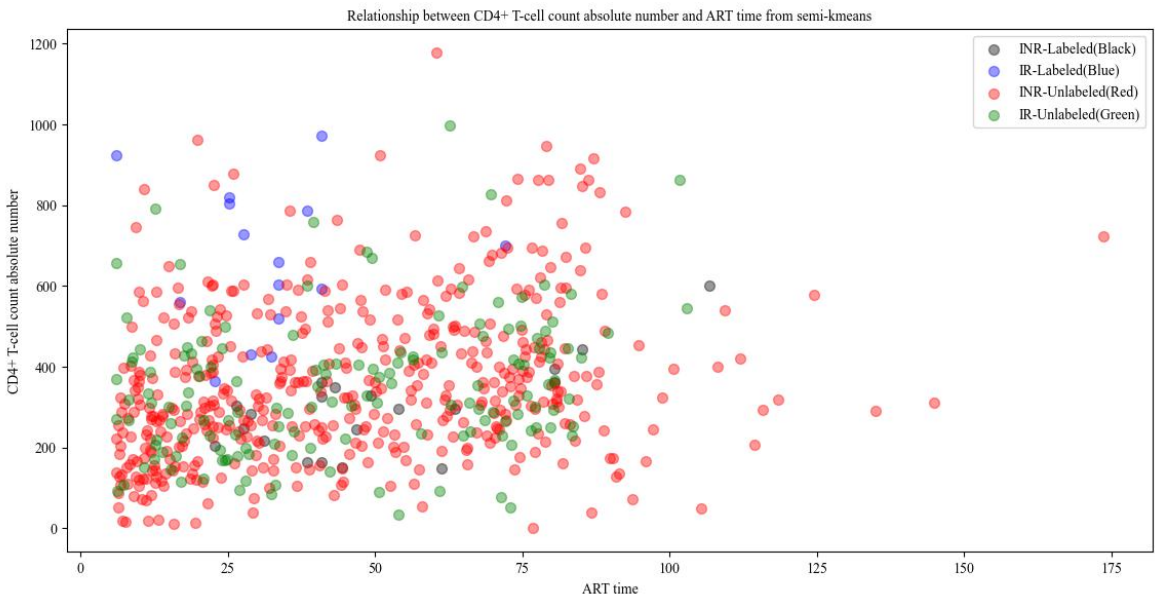

Figure 1 Clustering result from semi-kmeans

3.2.2 LabelPropagation

```
from sklearn.semi_supervised import LabelPropagation
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import metrics
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

df=pd.read_excel('inr_data.xlsx')
df=df.values.tolist()
df=np.array(df)

X0=df[:,:2]
Y0=df[:,2]

df_1=pd.read_excel('inr_own_1.xlsx')
df_1=df_1.values.tolist()
df_1=np.array(df_1)
X1=df_1
Y1=np.zeros(len(X1), dtype=float, order='C')-1

X=np.append(X0,X1,axis=0)
Y=np.append(Y0,Y1,axis=0)

x_trainval,x_test,y_trainval,y_test=train_test_split(X,Y,test_size=5,random_state=42)
```

```
x_train,x_val,y_train,y_val=train_test_split(x_trainval,y_trainval,test_size=5,random_state=42)
y_val_0=np.zeros(len(y_val), dtype=float, order='C')-1
y_trainval_0=np.append(y_train,y_val_0,axis=0)
y_test_0=np.zeros(len(y_test), dtype=float, order='C')-1
Y_0=np.append(y_trainval,y_test_0,axis=0)
# quit()
best_score=np.float64(0)
best_parameters={}

# knn
for max_iter in [10,100,1000,10000]:
    for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
        for n_neighbors in [3,5,7,9,11,13,15]:
            cls = LabelPropagation(max_iter=max_iter, kernel='knn', n_neighbors=n_neighbors)
            cls.fit(x_trainval, y_trainval_0)
            pred_result = cls.predict(x_trainval)
            score = metrics.accuracy_score(y_trainval, pred_result)
            if score > best_score:
                best_score = score
                best_parameters = {'tol': tol, 'n_neighbors': n_neighbors, 'max_iter':
max_iter,'kernel':'knn'}
                print("best para of knn")
                print(best_parameters)

# rbf
for max_iter in [10,100,1000,10000]:
    for tol in [0.01, 0.001, 0.0001, 0.00001, 0.000001]:
        for gamma in [100, 10, 1, 0.1, 0.01, 0.001, 0.0001]:
            cls1 = LabelPropagation(max_iter=max_iter, kernel='rbf', gamma=gamma)
            cls1.fit(x_trainval, y_trainval_0)
            pred_result = cls1.predict(x_trainval)
            score = metrics.accuracy_score(y_trainval, pred_result)
            if score > best_score:
                best_score = score
                best_parameters = {'tol': tol, 'gamma': gamma, 'max_iter': max_iter,'kernel':'rbf'}
                print("best para of rbf")
                print(best_parameters)
'''
best score on validation set :0.0480
best parameters:{'tol': 0.01, 'gamma': 0.001, 'max_iter': 10, 'kernel': 'rbf'}
'''
```

3.3 Modeling results analysis

For semi-kmeans, it can be seen from Figure 1 that the typing effect is very bad, and the data of INR and IR crosses very seriously. Therefore, semi-kmeans cannot be used to distinguish between INRs and IRs.

For LabelPropagation, in the process of screening the optimal parameters through cross-validation and grid search, we found that the optimal cross-validation parameters are'tol': 0.01,'gamma': 0.001,'max_iter': 10,'kernel': 'rbf', the best Cross-validation score is 0.0480. Since the accuracy rate of less than 5% is too low, LabelPropagation is not suitable for subsequent modeling and evaluation.

In summary, we did not use a semi-supervised model for modeling.

4.Unsupervised learning modeling

```
from sklearn.model_selection import    ShuffleSplit,cross_val_score
from sklearn.svm import SVC,LinearSVC
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
np.set_printoptions(threshold=np.inf)
import warnings
warnings.filterwarnings('ignore')

import pymssql
class    Dataoperator:
    def runsql_search(self):
        conn = pymssql.connect(server='127.0.0.1', port=1433, user='sa', password='alex0414',
                               database='AI_AIDS_Data')
        cursor_1 = conn.cursor()
        cursor_1.execute(self)

        return (cursor_1.fetchall())
    def runsql_insert(self,insertcontent):
        conn = pymssql.connect(server='127.0.0.1', port=1433, user='sa', password='alex0414',
                               database='AI_AIDS_Data')
        cursor=conn.cursor()
        cursor.executemany(self,insertcontent)
        conn.commit()
        conn.close()

def loadFeaturename_fromdatatable(sql):
    dt = Dataoperator.runsql_search(sql)
    if(dt==None or len(dt)==0):
        return None
    else:
        fr=dt
    return np.array(fr)
sql0="select * from inr_supvised where inr!=''"
df_1=loadFeaturename_fromdatatable(sql0)
```

```
df_1=np.array(df_1)
df_1=df_1.tolist()
df=pd.read_excel('inr_data.xlsx')
df=df.values.tolist()
df=np.array(df)
df=df.tolist()
df_total=np.append(df,df_1,axis=0)
df_total=np.array(df_total)

df0=df_total[:,:2]
df1=df_total[:,2]

from sklearn.cluster import KMeans
from sklearn.cluster import Birch
from sklearn.cluster import DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import MeanShift
from sklearn.cluster import OPTICS

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_scaler=scaler.fit_transform(df0)
x=x_scaler
kmeans=KMeans(n_clusters=2)
kmeans.fit(x)
ypred=kmeans.predict(x)

from sklearn.metrics import recall_score,f1_score,precision_score,accuracy_score,roc_auc_score
print("1-kmeans")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

agg=Birch(n_clusters=2)
agg.fit(x)
ypred=agg.fit_predict(x)
print("2-Birch")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
```

```
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

dbscan=DBSCAN(n_jobs=2)
dbscan.fit(x)
ypred=dbscan.fit_predict(x)
print("3-DBSCAN")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

agg=GaussianMixture(n_components=2)
agg.fit(x)
ypred=agg.fit_predict(x)
print("4-GaussianMixture")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

agg=AgglomerativeClustering(n_clusters=2)
agg.fit(x)
ypred=agg.fit_predict(x)
print("5-AgglomerativeClustering")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

agg=MeanShift(n_jobs=2)
agg.fit(x)
ypred=agg.fit_predict(x)
print("6-MeanShift")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

agg=OPTICS(n_jobs=2)
agg.fit(x)
```

```
ypred=agg.fit_predict(x)
print("7-OPTICS")
print("accuracy_score",accuracy_score(df1,ypred))
print("recall_score",recall_score(df1,ypred,average='micro'))
print("f1_score",f1_score(df1,ypred,average='micro'))
print("precision_score",precision_score(df1,ypred,average='micro'))
print("roc_auc_score",roc_auc_score(df1,ypred,average='micro'))

'''
Result
1-kmeans
accuracy_score 0.4500768049155146
recall_score 0.4500768049155146
f1_score 0.4500768049155146
precision_score 0.4500768049155146
roc_auc_score 0.45853758169934644
2-Birch
accuracy_score 0.7741935483870968
recall_score 0.7741935483870968
f1_score 0.7741935483870968
precision_score 0.7741935483870968
roc_auc_score 0.6171875
3-DBSCAN
accuracy_score 0.6912442396313364
recall_score 0.6912442396313364
f1_score 0.6912442396313364
precision_score 0.6912442396313364
roc_auc_score 0.4915747549019608
4-GaussianMixture
accuracy_score 0.5360983102918587
recall_score 0.5360983102918587
f1_score 0.5360983102918587
precision_score 0.5360983102918587
roc_auc_score 0.4710648148148148
5-AgglomerativeClustering
accuracy_score 0.47772657450076805
recall_score 0.47772657450076805
f1_score 0.47772657450076805
precision_score 0.47772657450076805
roc_auc_score 0.4311853213507625
'''
```

Since the unsupervised learning clustering accuracy was not high enough, and the results of the unsupervised learning clustering model were poor in interpretability, we gave up using the unsupervised learning model to define INR.

References

[1] Lu W, Feng YD, Jing FH, Han Y, et al. Association Between Gut Microbiota and CD4 Recovery in HIV-1 Infected Patients. Frontiers in microbiology. 2018 Jul 2;9:1451. https://doi.org/ 10.3389/fmicb.2018.01451.