In [1]:

```
import numpy as np
import pandas as pd
import random
import math,time,sys
from matplotlib import pyplot
from datetime import datetime
#from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
#==================================================================
def sigmoid1(gamma):     #convert to probability
        if gamma < 0:
                return 1 - 1/(1 + math.exp(gamma))
        else:
                return 1/(1 + math.exp(-gamma))

def sigmoid1i(gamma):     #convert to probability
        gamma = -gamma
        if gamma < 0:
                return 1 - 1/(1 + math.exp(gamma))
        else:
                return 1/(1 + math.exp(-gamma))

def sigmoid2(gamma):
        gamma /= 2
        if gamma < 0:
                return 1 - 1/(1 + math.exp(gamma))
        else:
                return 1/(1 + math.exp(-gamma))

def sigmoid3(gamma):
        gamma /= 3
        if gamma < 0:
                return 1 - 1/(1 + math.exp(gamma))
        else:
                return 1/(1 + math.exp(-gamma))

def sigmoid4(gamma):
        gamma *= 2
        if gamma < 0:
                return 1 - 1/(1 + math.exp(gamma))
        else:
                return 1/(1 + math.exp(-gamma))


def Vfunction1(gamma):
        return abs(np.tanh(gamma))

def Vfunction2(gamma):
        val = (math.pi)**(0.5)
        val /= 2
        val *= gamma
        val = math.erf(val)
        return abs(val)

def Vfunction3(gamma):
        val = 1 + gamma*gamma
        val = math.sqrt(val)
        val = gamma/val
```

```
        return abs(val)

def Vfunction4(gamma):
        val=(math.pi/2)*gamma
        val=np.arctan(val)
        val=(2/math.pi)*val
        return abs(val)

def initialize(popSize,dim):
        population=np.zeros((popSize,dim))
        minn = 1
        maxx = math.floor(0.8*dim)
        if maxx<minn:
                minn = maxx

        for i in range(popSize):
                random.seed(i**3 + 10 + time.time() )
                no = random.randint(minn,maxx)
                if no == 0:
                        no = 1
                random.seed(time.time()+ 100)
                pos = random.sample(range(0,dim-1),no)
                for j in pos:
                        population[i][j]=1

                # print(population[i])
        return population

def fitness(solution, trainX, testX, trainy, testy):
        cols=np.flatnonzero(solution)
        val=1
        if np.shape(cols)[0]==0:
                return val
        clf=RandomForestClassifier(n_estimators=10)
        train_data=trainX[:,cols]
        test_data=testX[:,cols]
        clf.fit(train_data,trainy)
        val=1-clf.score(test_data,testy)

        #in case of multi objective  []
        set_cnt=sum(solution)
        set_cnt=set_cnt/np.shape(solution)[0]
        val=omega*val+(1-omega)*set_cnt
        return val

def allfit(population, trainX, testX, trainy, testy):
        x=np.shape(population)[0]
        acc=np.zeros(x)
        for i in range(x):
                acc[i]=fitness(population[i],trainX,testX,trainy,testy)
                #print(acc[i])
        return acc

def toBinary(solution,dimension,trainX,testX,trainy,testy):
        # print("continuous",solution)

        Xnew = np.zeros(np.shape(solution))
        for i in range(dimension):
                temp = sigmoid1(solution[i])

                random.seed(time.time()+i)
```

```
                if temp > 0.5: # sfunction
                        Xnew[i] = 1
                else:
                        Xnew[i] = 0
                # if temp > 0.5: # vfunction
                #       Xnew[i] = 1 - solution[i]
                # else:
                #       Xnew[i] = solution[i]

        # Xnew = np.zeros(np.shape(solution))
        # Xnew1 = np.zeros(np.shape(solution))
        # Xnew2 = np.zeros(np.shape(solution))
        # for i in range(dimension):
        #       temp = sigmoid1(abs(solution[i]))
        #       random.seed(time.time()+i)
        #       r1 = random.random()
        #       if temp > r1: # sfunction
        #               Xnew1[i] = 1
        #       else:
        #               Xnew1[i] = 0

        #       temp = sigmoid1i(abs(solution[i]))
        #       if temp > r1: # sfunction
        #               Xnew2[i] = 1
        #       else:
        #               Xnew2[i] = 0

        # fit1 = fitness(Xnew1,trainX,testX,trainy,testy)
        # fit2 = fitness(Xnew2,trainX,testX,trainy,testy)
        # fitOld =  fitness(solution,trainX,testX,trainy,testy)
        # if fit1<fitOld or fit2<fitOld:
        #       if fit1 < fit2:
        #               Xnew = Xnew1.copy()
        #       else:
        #               Xnew = Xnew2.copy()
        # return Xnew
        # # else: CROSSOVER
        # Xnew3 = Xnew1.copy()
        # Xnew4 = Xnew2.copy()
        # for i in range(dimension):
        #       random.seed(time.time() + i)
        #       r2 = random.random()
        #       if r2>0.5:
        #               tx = Xnew3[i]
        #               Xnew3[i] = Xnew4[i]
        #               Xnew4[i] = tx
        # fit1 = fitness(Xnew3,trainX,testX,trainy,testy)
        # fit2 = fitness(Xnew4,trainX,testX,trainy,testy)
        # if fit1<fit2:
        #       return Xnew3
        # else:
        #       return Xnew4
        # print("binary",Xnew)
        return Xnew


#============================================================================
def socialmimic(dataset,popSize,maxIter):

        #--------------------------------------------------------------------
        df=pd.read_csv(dataset)
```

```
        (a,b)=np.shape(df)
        #print(a,b)
        data = df.values[:,0:b-1]
        label = df.values[:,b-1]
        dimension = np.shape(data)[1] #solution dimension
        #----------------------------------------------------------------------

        cross = 5
        test_size = (1/cross)
        trainX, testX, trainy, testy = train_test_split(data, label,stratify=label ,tes
t_size=test_size)


        clf=RandomForestClassifier(n_estimators=10)
        clf.fit(trainX,trainy)
        val=clf.score(testX,testy)
        whole_accuracy = val
        #print("Total Acc: ",val)

        x_axis = []
        y_axis = []
        population = initialize(popSize,dimension)
        GBESTSOL = np.zeros(np.shape(population[0]))
        GBESTFIT = 1000

        start_time = datetime.now()

        for currIter in range(1,maxIter):
                newpop = np.zeros((popSize,dimension))
                fitList = allfit(population,trainX,testX,trainy,testy)
                y_axis.append(min(fitList))
                x_axis.append(currIter)
                bestInx = np.argmin(fitList)
                fitBest = min(fitList)
                Xbest = population[bestInx].copy()

                if fitBest<GBESTFIT:
                        GBESTFIT = fitBest
                        GBESTSOL = Xbest.copy()
                        #print("",GBESTSOL.sum())

                for i in range(popSize):
                        currFit = fitList[i]
                        # print(currFit)
                        difference = ( currFit - fitBest ) / currFit
                        if difference == 0:
                                random.seed(time.time())
                                difference = random.uniform(0,1)
                        newpop[i] = np.add(population[i],np.multiply(difference,populat
ion[i]))
                        newpop[i] = toBinary(population[i],dimension,trainX,testX,train
y,testy)

                population = newpop.copy()
        # pyplot.plot(x_axis,y_axis)
        # pyplot.show()

        #test accuracy
        cols = np.flatnonzero(GBESTSOL)
        val = 1
        if np.shape(cols)[0]==0:
```

```
                return GBESTSOL
        clf = RandomForestClassifier(n_estimators=10)
        train_data = trainX[:,cols]
        test_data = testX[:,cols]
        clf.fit(train_data,trainy)
        val = clf.score(test_data,testy)
        return GBESTSOL,val


#=======================================================================================
==================
popSize = 20
maxIter = 10
omega = 0.9
datasetList = ["FinalData_3000"]
# datasetList = ["Breastcancer", "BreastEW", "CongressEW", "Exactly", "Exactly2", "Hear
tEW", "Ionosphere", #"KrVsKpEW","Lymphography", "M-of-n", "PenglungEW", "Sonar", "Spect
EW", "Tic-tac-toe", "Vote", "WaveformEW", "Wine", "Zoo"]

for dataset in datasetList:
        accuList = []
        featList = []
        for count in range(3001):
                if (dataset == "FinalData_3000") and count>1500:
                        break
                #print(count)
                answer,testAcc = socialmimic("C:/Users/IYI/Desktop/matlab_yedek/aaa_min
eral/minerals/birlestirilmis_3000/"+dataset+".csv",popSize,maxIter)
                print(answer.sum())
                accuList.append(testAcc)
                #print(featList.append(answer.sum()))
        inx = np.argmax(accuList)
        best_accuracy = accuList[inx]
        best_no_features = featList[inx]
        print(dataset,"best:",accuList[inx],featList[inx])

        with open("C:/Users/IYI/Desktop/matlab_yedek/aaa_mineral/minerals/birlestirilmi
s_3000/result_FinalData2.csv","a") as f:
                print(dataset,"%.2f" % (100*best_accuracy),best_no_features,file=f)
```

```
1848.0
629.0
764.0
1185.0
981.0
396.0
1192.0
1820.0
1588.0
836.0
1072.0
1065.0
1000.0
1090.0
1031.0
1586.0
1037.0
493.0
1179.0
1758.0
1136.0
1719.0
987.0
802.0
1558.0
533.0
1120.0
1769.0
818.0
740.0
785.0
1364.0
681.0
583.0
1152.0
936.0
1746.0
570.0
888.0
533.0
657.0
1572.0
2068.0
1139.0
1041.0
1914.0
350.0
365.0
972.0
371.0
195.0
1723.0
1008.0
910.0
1365.0
337.0
926.0
279.0
470.0
602.0
579.0
```

```
580.0
1342.0
1542.0
671.0
973.0
1077.0
1438.0
518.0
659.0
708.0
1282.0
524.0
1253.0
1285.0
1757.0
1804.0
414.0
974.0
1323.0
649.0
1208.0
1375.0
937.0
633.0
955.0
1830.0
1240.0
525.0
253.0
993.0
1386.0
1319.0
420.0
1240.0
1468.0
2336.0
735.0
2282.0
729.0
526.0
516.0
495.0
1846.0
1202.0
368.0
1627.0
292.0
447.0
715.0
929.0
1124.0
966.0
694.0
1879.0
795.0
1080.0
550.0
533.0
755.0
1870.0
1737.0
```

```
630.0
234.0
589.0
802.0
1741.0
1079.0
998.0
1046.0
1033.0
354.0
1310.0
805.0
1330.0
556.0
672.0
179.0
1201.0
730.0
551.0
449.0
897.0
1124.0
882.0
402.0
852.0
382.0
347.0
197.0
968.0
861.0
1078.0
1297.0
707.0
1202.0
380.0
722.0
737.0
1771.0
1372.0
1000.0
1242.0
604.0
1611.0
424.0
625.0
1465.0
779.0
882.0
866.0
644.0
477.0
136.0
230.0
712.0
1362.0
942.0
750.0
965.0
1033.0
535.0
1596.0
```

```
972.0
933.0
811.0
963.0
909.0
1114.0
678.0
377.0
940.0
466.0
659.0
1069.0
2140.0
1663.0
1631.0
982.0
644.0
1045.0
845.0
565.0
779.0
899.0
1792.0
1372.0
1039.0
1736.0
878.0
573.0
452.0
955.0
1775.0
1034.0
1682.0
840.0
998.0
754.0
499.0
710.0
407.0
1347.0
969.0
1470.0
849.0
879.0
464.0
1043.0
354.0
432.0
1616.0
1083.0
346.0
1639.0
2322.0
326.0
1132.0
711.0
312.0
488.0
968.0
1571.0
291.0
```

```
1253.0
586.0
593.0
554.0
320.0
918.0
822.0
447.0
683.0
1860.0
141.0
1032.0
1336.0
1925.0
1350.0
1262.0
959.0
661.0
1595.0
613.0
1137.0
973.0
920.0
1152.0
1000.0
1219.0
832.0
1106.0
1458.0
951.0
1057.0
907.0
463.0
1912.0
367.0
1496.0
760.0
1873.0
895.0
871.0
747.0
581.0
1794.0
1246.0
1591.0
594.0
431.0
228.0
318.0
1058.0
865.0
982.0
1143.0
775.0
1739.0
1755.0
597.0
979.0
489.0
547.0
1302.0
```

```
823.0
2054.0
1307.0
329.0
605.0
414.0
1519.0
962.0
320.0
714.0
676.0
736.0
429.0
509.0
1432.0
1124.0
549.0
777.0
451.0
814.0
341.0
710.0
1319.0
768.0
584.0
1347.0
1722.0
698.0
1320.0
1442.0
783.0
1117.0
1958.0
536.0
602.0
901.0
630.0
657.0
686.0
400.0
1160.0
1109.0
1295.0
280.0
665.0
1873.0
1200.0
338.0
778.0
840.0
1499.0
2277.0
381.0
1264.0
350.0
2152.0
595.0
583.0
1200.0
1048.0
872.0
```

```
648.0
535.0
1281.0
1379.0
407.0
849.0
1345.0
803.0
817.0
846.0
403.0
662.0
765.0
996.0
1554.0
991.0
864.0
1363.0
538.0
1148.0
232.0
372.0
557.0
550.0
602.0
512.0
798.0
813.0
418.0
1891.0
1600.0
1143.0
534.0
194.0
831.0
1462.0
539.0
1601.0
968.0
668.0
533.0
589.0
615.0
1066.0
609.0
1075.0
782.0
790.0
477.0
955.0
1626.0
1186.0
264.0
809.0
882.0
1153.0
1249.0
1241.0
357.0
970.0
1231.0
```

```
323.0
1058.0
2020.0
753.0
497.0
1012.0
680.0
673.0
1090.0
1167.0
319.0
740.0
473.0
1421.0
779.0
641.0
1088.0
929.0
732.0
1378.0
810.0
1218.0
1191.0
610.0
177.0
500.0
295.0
509.0
1769.0
1025.0
1163.0
2361.0
1458.0
1649.0
611.0
834.0
292.0
818.0
739.0
203.0
1970.0
1365.0
405.0
532.0
1689.0
619.0
1201.0
603.0
1813.0
1416.0
772.0
1141.0
567.0
1573.0
1501.0
330.0
973.0
1264.0
361.0
556.0
1672.0
```

```
1285.0
625.0
290.0
1559.0
930.0
1205.0
800.0
779.0
1030.0
1647.0
1043.0
798.0
1274.0
765.0
931.0
841.0
587.0
501.0
427.0
875.0
458.0
242.0
582.0
1898.0
1242.0
1226.0
1093.0
1051.0
1153.0
1612.0
790.0
569.0
401.0
1310.0
990.0
599.0
1154.0
1432.0
978.0
1506.0
894.0
772.0
283.0
685.0
1705.0
1807.0
1995.0
2224.0
731.0
894.0
1165.0
631.0
231.0
1117.0
788.0
1871.0
422.0
926.0
445.0
779.0
670.0
```

```
1962.0
869.0
1129.0
1317.0
659.0
1035.0
170.0
489.0
560.0
1353.0
2009.0
808.0
733.0
922.0
986.0
604.0
596.0
681.0
1985.0
523.0
1569.0
709.0
968.0
764.0
934.0
2030.0
1421.0
2310.0
812.0
966.0
1242.0
1421.0
471.0
312.0
828.0
932.0
1049.0
340.0
410.0
1272.0
1404.0
1052.0
1383.0
428.0
492.0
661.0
435.0
365.0
513.0
882.0
880.0
374.0
666.0
497.0
1150.0
916.0
907.0
1069.0
1024.0
1145.0
469.0
```

```
746.0
1286.0
391.0
331.0
1275.0
630.0
628.0
1356.0
967.0
1184.0
605.0
358.0
1092.0
1322.0
1108.0
1124.0
606.0
1004.0
219.0
640.0
732.0
876.0
760.0
829.0
769.0
291.0
786.0
1013.0
1447.0
918.0
846.0
861.0
1761.0
1760.0
528.0
613.0
889.0
499.0
575.0
1316.0
824.0
2212.0
1381.0
798.0
734.0
529.0
481.0
1273.0
980.0
714.0
602.0
1088.0
780.0
325.0
542.0
1287.0
841.0
1225.0
2048.0
1574.0
823.0
```

```
172.0
774.0
700.0
726.0
435.0
361.0
1173.0
1693.0
586.0
957.0
501.0
456.0
1369.0
219.0
632.0
859.0
277.0
1001.0
1089.0
343.0
961.0
1781.0
1786.0
619.0
691.0
486.0
693.0
810.0
597.0
474.0
487.0
747.0
430.0
875.0
1597.0
1665.0
612.0
598.0
488.0
1001.0
437.0
979.0
679.0
1513.0
708.0
816.0
502.0
220.0
1019.0
248.0
749.0
596.0
1160.0
517.0
960.0
1078.0
272.0
721.0
1124.0
2028.0
823.0
```

```
225.0
690.0
1018.0
1573.0
1240.0
884.0
801.0
638.0
1316.0
342.0
1560.0
1239.0
500.0
1170.0
623.0
1123.0
844.0
558.0
454.0
580.0
1697.0
1291.0
702.0
831.0
230.0
163.0
293.0
1671.0
2212.0
1250.0
1196.0
990.0
683.0
731.0
1347.0
391.0
739.0
932.0
1396.0
604.0
579.0
1488.0
1179.0
1690.0
1748.0
163.0
2294.0
910.0
637.0
493.0
1372.0
1076.0
530.0
1384.0
1504.0
137.0
1012.0
766.0
489.0
942.0
1036.0
```

```
946.0
835.0
1637.0
1004.0
869.0
662.0
490.0
605.0
1475.0
784.0
1313.0
528.0
789.0
1113.0
995.0
338.0
1613.0
1864.0
507.0
862.0
846.0
1916.0
306.0
416.0
1811.0
1923.0
1118.0
957.0
1626.0
571.0
602.0
515.0
551.0
596.0
502.0
2116.0
1715.0
1924.0
647.0
1552.0
1202.0
785.0
1149.0
993.0
231.0
291.0
667.0
473.0
1190.0
396.0
1128.0
664.0
1111.0
1531.0
651.0
1194.0
968.0
1088.0
541.0
1464.0
1210.0
```

```
350.0
1216.0
526.0
696.0
635.0
1387.0
348.0
1501.0
229.0
782.0
1400.0
472.0
1258.0
1508.0
208.0
1279.0
455.0
159.0
291.0
1732.0
462.0
1552.0
813.0
1007.0
786.0
1837.0
717.0
1745.0
555.0
816.0
1532.0
1121.0
1032.0
724.0
1754.0
589.0
723.0
861.0
317.0
559.0
731.0
1429.0
926.0
1111.0
944.0
776.0
2041.0
595.0
1110.0
1733.0
862.0
736.0
1187.0
1010.0
1160.0
987.0
812.0
589.0
376.0
1252.0
1384.0
```

```
1348.0
302.0
2255.0
1013.0
1006.0
649.0
886.0
733.0
1173.0
608.0
908.0
670.0
1658.0
137.0
930.0
1061.0
522.0
2256.0
396.0
446.0
1909.0
542.0
1318.0
1738.0
347.0
354.0
1318.0
1392.0
617.0
979.0
1522.0
423.0
738.0
1157.0
1370.0
1685.0
308.0
681.0
769.0
634.0
1102.0
599.0
764.0
383.0
1669.0
491.0
226.0
908.0
954.0
556.0
540.0
1118.0
221.0
1275.0
1040.0
238.0
647.0
1354.0
936.0
1297.0
190.0
```

```
1498.0
1535.0
1237.0
1075.0
1076.0
318.0
512.0
1191.0
1488.0
1751.0
215.0
1041.0
657.0
1402.0
1291.0
499.0
744.0
942.0
726.0
568.0
1238.0
1283.0
838.0
1164.0
797.0
699.0
1241.0
391.0
988.0
424.0
815.0
1366.0
1855.0
587.0
648.0
422.0
434.0
835.0
2075.0
758.0
934.0
324.0
1304.0
1369.0
818.0
950.0
626.0
910.0
245.0
541.0
482.0
524.0
530.0
629.0
953.0
1067.0
880.0
1851.0
383.0
506.0
896.0
```

```
484.0
1855.0
475.0
1059.0
794.0
741.0
1120.0
951.0
881.0
858.0
372.0
950.0
953.0
1551.0
1033.0
639.0
1120.0
487.0
290.0
1038.0
968.0
1429.0
1317.0
1596.0
149.0
1679.0
1059.0
757.0
699.0
121.0
271.0
884.0
335.0
1903.0
617.0
430.0
324.0
1100.0
1172.0
749.0
961.0
696.0
972.0
2208.0
1139.0
895.0
1387.0
599.0
2312.0
1296.0
1402.0
734.0
783.0
1008.0
1718.0
336.0
386.0
919.0
1486.0
382.0
770.0
```

```
849.0
236.0
344.0
680.0
1063.0
296.0
374.0
1072.0
1036.0
1747.0
1275.0
1560.0
682.0
626.0
279.0
677.0
1749.0
596.0
482.0
453.0
1674.0
1525.0
850.0
1261.0
805.0
599.0
696.0
1022.0
1089.0
457.0
754.0
1068.0
1737.0
760.0
512.0
564.0
520.0
634.0
911.0
435.0
349.0
491.0
539.0
229.0
1090.0
944.0
1012.0
1212.0
1255.0
857.0
1043.0
330.0
1389.0
644.0
860.0
887.0
1338.0
469.0
1415.0
1290.0
1692.0
```

```
435.0
599.0
1875.0
947.0
918.0
1284.0
775.0
1884.0
1602.0
873.0
1163.0
738.0
1345.0
561.0
745.0
462.0
1105.0
584.0
504.0
592.0
218.0
442.0
1216.0
1450.0
730.0
564.0
598.0
868.0
1631.0
209.0
1268.0
1466.0
429.0
1579.0
1971.0
993.0
374.0
1306.0
470.0
826.0
484.0
1306.0
474.0
425.0
325.0
178.0
350.0
1154.0
814.0
698.0
615.0
804.0
517.0
1444.0
1065.0
1235.0
988.0
598.0
1647.0
600.0
764.0
```

```
145.0
2261.0
733.0
413.0
845.0
1028.0
1552.0
915.0
479.0
1404.0
1433.0
1814.0
2075.0
1370.0
1785.0
945.0
466.0
565.0
844.0
948.0
1473.0
1537.0
477.0
1923.0
506.0
1147.0
1684.0
932.0
275.0
1160.0
410.0
503.0
1710.0
719.0
335.0
846.0
500.0
716.0
512.0
1160.0
972.0
1820.0
1188.0
692.0
727.0
796.0
1089.0
678.0
452.0
1070.0
367.0
625.0
1153.0
990.0
1870.0
557.0
352.0
1154.0
1694.0
390.0
310.0
```

```
798.0
579.0
471.0
1592.0
599.0
1201.0
1055.0
634.0
680.0
1090.0
329.0
381.0
1637.0
841.0
1675.0
1341.0
486.0
658.0
833.0
624.0
1325.0
392.0
874.0
1005.0
408.0
1106.0
685.0
915.0
1459.0
680.0
529.0
715.0
1394.0
1023.0
472.0
1544.0
733.0
802.0
317.0
1170.0
997.0
537.0
953.0
458.0
1220.0
2200.0
869.0
1385.0
302.0
396.0
875.0
763.0
750.0
934.0
579.0
984.0
386.0
938.0
961.0
1666.0
1244.0
```

```
941.0
967.0
516.0
1736.0
510.0
1733.0
413.0
500.0
449.0
1561.0
1449.0
1010.0
963.0
596.0
1269.0
360.0
616.0
477.0
276.0
1865.0
1622.0
1202.0
531.0
339.0
772.0
1227.0
942.0
500.0
863.0
586.0
1036.0
795.0
1204.0
1565.0
1528.0
837.0
1518.0
1480.0
1120.0
184.0
1140.0
2050.0
1352.0
1384.0
1805.0
715.0
1441.0
191.0
424.0
2275.0
611.0
600.0
1024.0
352.0
201.0
2125.0
636.0
704.0
183.0
674.0
519.0
```

```
880.0
912.0
1411.0
1181.0
1310.0
356.0
1074.0
208.0
414.0
750.0
1306.0
304.0
1178.0
491.0
2390.0
980.0
938.0
1314.0
1177.0
283.0
424.0
431.0
421.0
576.0
981.0
854.0
956.0
738.0
684.0
830.0
692.0
285.0
675.0
1465.0
260.0
919.0
1408.0
652.0
309.0
816.0
1487.0
942.0
317.0
1496.0
605.0
721.0
873.0
940.0
915.0
994.0
512.0
299.0
703.0
427.0
1236.0
582.0
779.0
453.0
453.0
1394.0
502.0
```

```
738.0
882.0
1049.0
230.0
1194.0
1015.0
511.0
219.0
688.0
1585.0
1036.0
2041.0
1342.0
771.0
1469.0
838.0
1746.0
383.0
1155.0
285.0
1106.0
595.0
666.0
735.0
2308.0
533.0
763.0
1416.0
841.0
978.0
815.0
1650.0
1200.0
1957.0
1875.0
1116.0
526.0

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-1-847a4068c976> in <module>()
    267         inx = np.argmax(accuList)
    268         best_accuracy = accuList[inx]
--> 269         best_no_features = featList[inx]
    270         print(dataset,"best:",accuList[inx],featList[inx])
    271 

IndexError: list index out of range
```