In [ ]:

```
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.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier


##########################################################################################
#########################3
def sigmoid1(gamma):     #convert to probability
        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 fitness(position):
        cols=np.flatnonzero(position)
        val=1
        if np.shape(cols)[0]==0:
                return val
        clf = RandomForestClassifier(n_estimators=10)
        #clf=KNeighborsClassifier(n_neighbors=5)
        # clf=MLPClassifier( alpha=0.01, max_iter=1000) #hidden_layer_sizes=(1000,500,1
00)
        #cross=3
        #test_size=(1/cross)
        #X_train, X_test, y_train, y_test = train_test_split(trainX, trainy,  stratify=
trainy,test_size=test_size)
        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(position)
        set_cnt=set_cnt/np.shape(position)[0]
        val=omega*val+(1-omega)*set_cnt
        return val

def onecount(position):
        cnt=0
        for i in position:
                if i==1.0:
                        cnt+=1
        return cnt


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

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 toBinary(population,popSize,dimension,oldPop):

        for i in range(popSize):
                for j in range(dimension):
                        temp = Vfunction3(population[i][j])

                        # if temp > 0.5: # sfunction
                        #       population[i][j] = 1
                        # else:
                        #       population[i][j] = 0

                        if temp > 0.5: # vfunction
                                population[i][j] = (1 - oldPop[i][j])
                        else:
                                population[i][j] = oldPop[i][j]
        return population


##############################################################################################
omega = 0.9 #weightage for no of features and accuracy
popSize = 20
max_iter = 10
S = 2


# df=pd.read_csv(sys.argv[1])
# (a,b)=np.shape(df)
# print(a,b)
# data = df.values[:,0:b-1]
# label = df.values[:,b-1]
# dimension = np.shape(data)[1] #particle dimension


best_accuracy = -1
best_no_features = -1
average_accuracy = 0
global_count = 0
accuracy_list = []
features_list = []

for train_iteration in range(1500):

        #---------------------------------------------------------------------
        #I know I should not put not it here, but still ...
        df=pd.read_csv("C:/Users/IYI/Desktop/matlab_yedek/aaa_mineral/minerals/birlesti
rilmis_3000/FinalData_3000.csv")
        (a,b)=np.shape(df)
        #print(a,b)
        data = df.values[:,0:b-1]
        label = df.values[:,b-1]
        dimension = np.shape(data)[1] #particle 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=KNeighborsClassifier(n_neighbors=5)
        # clf=MLPClassifier(alpha=0.001, max_iter=1000) #hidden_layer_sizes=(1000,500,1
00)
        clf.fit(trainX,trainy)
        val=clf.score(testX,testy)
        whole_accuracy = val
        #print("Total Acc: ",val)

        # for population_iteration in range(2):
        global_count += 1
        #print('global: ',global_count)

        x_axis = []
        y_axis = []

        population = initialize(popSize,dimension)
        # print(population)

        start_time = datetime.now()
        fitList = allfit(population)
        bestInx = np.argmin(fitList)
        fitBest = min(fitList)
        Mbest = population[bestInx].copy()
        for currIter in range(max_iter):

                popnew = np.zeros((popSize,dimension))
                x_axis.append(currIter)
                y_axis.append(min(fitList))
                for i in range(popSize):
                        random.seed(time.time() + 10.01)
                        randNo = random.random()
                        if randNo<0.5 :
                                #chain foraging
                                random.seed(time.time())
                                r = random.random()
                                alpha = 2*r*(abs(math.log(r))**0.5)
                                if i == 1:
                                        popnew[i] = population[i] + r * (Mbest - popula
tion[i]) + alpha*(Mbest - population[i])
                                else:
                                        popnew[i] = population[i] + r * (population[i-1
] - population[i]) + alpha*(Mbest - population[i])
                        else:
                                #cyclone foraging
                                cutOff = random.random()
                                r = random.random()
                                r1 = random.random()
                                beta = 2 * math.exp(r1 * (max_iter - currIter + 1) / ma
x_iter) * math.sin(2 * math.pi * r1)
                                if currIter/max_iter < cutOff:
                                        # exploration
                                        Mrand = np.zeros(np.shape(population[0]))
                                        no = random.randint(1,max(int(0.1*dimension),2
))
                                        random.seed(time.time()+ 100)
                                        pos = random.sample(range(0,dimension-1),no)
```

```
                                        for j in pos:
                                                Mrand[j] = 1

                                        if i==1 :
                                                popnew[i] = Mrand + r * (Mrand - popula
tion[i]) + beta * (Mrand - population[i])
                                        else:
                                                popnew[i] = Mrand + r * (population[i-1
] - population[i]) + beta * (Mrand - population[i])
                                else:
                                        # exploitation
                                        if i == 1:
                                                popnew[i] = Mbest + r * (Mbest - popula
tion[i]) + beta * (Mbest - population[i])
                                        else:
                                                popnew[i] = Mbest + r * (population[i-1
] - population[i]) + beta * (Mbest - population[i])

                # print(popnew)

                popnew = toBinary(popnew,popSize,dimension,population)
                popnewTemp = popnew.copy()
                #compute fitness for each individual
                fitList = allfit(popnew)
                if min(fitList)<fitBest :
                        bestInx = np.argmin(fitList)
                        fitBest = min(fitList)
                        Mbest = popnew[bestInx].copy()
                # print(fitList,fitBest)

                #somersault foraging
                for i in range(popSize):
                        r2 = random.random()
                        random.seed(time.time())
                        r3 = random.random()
                        popnew[i] = popnew[i] + S * (r2*Mbest - r3*popnew[i])

                popnew = toBinary(popnew,popSize,dimension,popnewTemp)
                #compute fitness for each individual
                fitList = allfit(popnew)
                if min(fitList)<fitBest :
                        bestInx = np.argmin(fitList)
                        fitBest = min(fitList)
                        Mbest = popnew[bestInx].copy()
                # print(fitList,fitBest)

                population = popnew.copy()


        time_required = datetime.now() - start_time

        # pyplot.plot(x_axis,y_axis)
        # pyplot.xlim(0,max_iter)
        # pyplot.ylim(max(0,min(y_axis)-0.1),min(max(y_axis)+0.1,1))
        # pyplot.show()


        output = Mbest.copy()
        #print(output)

        #test accuracy
```

```
        cols=np.flatnonzero(output)
        #print(cols)
        X_test=testX[:,cols]
        X_train=trainX[:,cols]
        #print(np.shape(feature))

        clf = RandomForestClassifier(n_estimators=10)
        #clf=KNeighborsClassifier(n_neighbors=5)
        #clf=MLPClassifier( alpha=0.001, max_iter=2000) #hidden_layer_sizes=(1000,500,1
00 ),
        clf.fit(X_train,trainy)
        val=clf.score(X_test, testy )


        accuracy_list.append(val)
        features_list.append(onecount(output))
        if ( val == best_accuracy ) and ( onecount(output) < best_no_features ):
                best_accuracy = val
                best_no_features = onecount( output )
                best_time_req = time_required
                best_whole_accuracy = whole_accuracy
                #print(onecount(output))
        if val > best_accuracy :
                best_accuracy = val
                best_no_features = onecount( output )
                best_time_req = time_required
                best_whole_accuracy = whole_accuracy
                #print(onecount(output))
        print(onecount(output))

print('', best_no_features)
# print('avg: ',average_accuracy/10)


# accuracy_list = np.array(accuracy_list)
# accuracy_list.sort()
# accuracy_list = accuracy_list[-4:]
# average = np.mean(accuracy_list)
# stddev = np.std(accuracy_list)

# accuracy_list = list(accuracy_list)
# avgFea = 0
# for i in accuracy_list:
#       inx = accuracy_list.index(i)
#       avgFea += features_list[inx]
# avgFea /= 4

temp=sys.argv[1].split('/')[-1]
with open("C:/Users/IYI/Desktop/matlab_yedek/aaa_mineral/minerals/birlestirilmis_3000/F
inalData_3001.csv","a") as f:
        print(temp,"%.2f" % (100*best_whole_accuracy) ,
                np.shape(df)[1] - 1,"%.2f" % (100*best_accuracy),best_no_features,file=
f)
```

```
639
254
160
675
983
1581
173
674
1922
2599
563
1846
835
892
437
964
633
864
535
1389
447
561
1093
1974
468
692
788
225
1295
855
321
795
753
1210
140
417
508
344
737
793
873
238
271
1865
396
1139
886
528
1117
849
521
382
233
1064
449
896
531
1013
1879
402
1265
```

```
111
591
336
872
216
1304
2367
709
1954
752
1086
234
314
2217
1150
1027
389
1784
258
843
465
641
776
1105
1416
390
776
1360
1600
209
1125
1270
391
977
769
1525
857
447
440
154
2000
1672
359
1069
855
2146
1154
529
920
483
353
366
1723
1926
1265
1806
701
1958
777
923
850
```

```
1241
429
1185
1900
195
450
1528
235
962
778
820
173
2052
373
830
471
891
2365
375
741
697
595
840
355
1604
562
439
866
1141
642
1118
851
1069
1750
984
491
1805
666
729
803
552
1154
852
740
710
626
910
1585
1488
976
303
471
367
1080
1008
868
422
584
799
548
918
```

```
2067
1422
909
1049
1260
368
1518
289
361
180
1144
537
1598
189
1603
1478
1120
820
412
1073
591
957
891
671
1633
1533
532
682
1518
645
1080
1266
990
1104
1194
946
1325
528
1397
880
434
172
1003
932
315
910
445
903
756
757
865
762
430
98
1110
240
747
138
406
576
988
```

```
658
165
731
731
289
153
1973
349
1068
580
2006
939
1112
724
1652
382
489
1004
817
1390
1208
677
750
496
1026
227
2055
939
1904
1517
296
383
1402
243
798
604
453
980
458
1597
983
170
952
619
941
1043
771
2414
191
942
1480
851
1189
997
1332
1175
399
745
1049
183
1421
```

```
424
1437
419
1444
150
580
1062
687
1071
298
292
452
1542
1069
1952
1536
1481
399
2232
1485
614
898
985
188
256
727
382
1211
902
529
305
1271
575
954
1173
650
286
398
1318
850
952
514
385
640
683
867
966
888
678
466
1628
1482
1062
346
456
739
669
689
799
982
969
```

```
762
1520
1021
105
1015
608
788
306
343
1513
691
821
932
1501
787
213
539
227
234
1545
685
288
61
313
1913
796
683
1439
893
1491
1569
648
290
664
896
499
1523
602
833
1771
1687
348
487
782
822
486
392
963
1182
876
723
563
352
351
473
1620
642
206
312
875
336
```

```
349
1154
811
1305
410
1446
1418
228
173
1362
782
2048
399
1768
516
598
1204
1500
2027
1244
769
484
1266
866
676
145
1110
231
458
394
1754
426
390
1515
737
541
1151
721
922
747
980
755
921
537
1562
486
430
395
1517
2021
1415
319
399
1975
521
1432
1589
1154
698
1063
1909
```

```
863
224
574
1388
643
1975
839
1096
1442
1218
481
489
1128
812
116
522
1342
894
679
1370
286
982
1761
1663
1463
1352
2175
697
590
398
344
312
1020
384
1570
535
553
672
1018
710
392
1575
2028
1388
981
612
818
1169
932
300
853
271
416
1201
807
1074
1081
377
396
689
468
```

```
1702
140
1154
785
936
787
499
1366
948
1132
733
1205
1830
470
645
1370
659
514
652
578
1025
673
1555
79
790
155
633
1306
254
1657
1261
151
1033
1079
688
862
1853
623
654
227
668
440
567
1012
1604
1366
1255
1326
759
1421
496
1452
261
800
1174
1228
2393
440
525
263
475
```

```
777
738
386
552
1293
601
1309
549
1381
890
729
245
588
1719
605
390
1052
889
399
1405
150
1018
1184
548
942
1620
182
1113
1309
1793
233
277
801
1588
1508
755
464
410
1412
454
836
930
694
192
175
1146
630
1492
797
598
182
604
2013
510
770
296
492
1710
908
1200
1044
```

```
700
1008
2236
256
1031
1161
887
1362
528
373
324
621
699
1992
237
588
1870
1386
1182
1026
1242
393
1312
645
526
301
1559
761
590
441
462
1395
1435
774
229
133
1845
924
855
1690
1370
1328
1268
416
371
1332
556
776
649
427
509
1410
158
703
886
603
768
280
1349
303
1288
```

```
749
2070
1291
1863
630
479
984
1873
1060
1189
834
693
325
1188
419
1043
725
908
1297
235
1121
585
1301
349
938
1259
587
671
607
2043
1593
399
108
390
1050
631
154
888
992
1079
459
1798
643
467
1365
249
268
1724
166
531
159
871
674
1072
1008
1000
1425
293
1121
820
157
```

```
256
740
1110
1165
177
651
1661
1047
696
1214
1466
2182
704
1052
407
1397
686
623
361
165
523
263
1098
344
725
273
281
201
539
242
518
472
1065
1451
1493
1528
1202
863
407
321
240
687
723
507
1052
1346
584
1104
743
1589
236
751
380
1016
1251
792
229
1354
469
1582
786
```

```
1449
321
1294
1716
1761
742
672
1126
407
746
535
1556
835
1736
803
2173
2302
667
432
680
515
1388
546
1288
183
1561
497
1681
760
1062
418
1456
255
621
1815
1570
671
1153
633
1318
253
362
750
61
1207
1066
1097
1451
581
1445
505
437
427
1009
1124
695
805
649
670
1909
759
```

```
1084
1173
1400
897
509
1062
1536
933
2144
747
820
965
1032
210
413
248
1489
878
1381
1009
538
317
1089
1149
1398
1010
676
691
2135
1636
1103
1988
642
762
1134
1499
639
858
876
541
1411
1442
929
356
554
1774
375
476
149
1540
1925
574
755
540
865
1387
121
656
426
1730
445
```

```
1171
566
1112
1146
1586
1014
667
117
296
407
921
843
228
1332
847
2218
411
422
328
300
1192
1035
1913
2347
636
497
177
806
1260
183
336
380
671
982
1701
587
566
1249
425
442
384
750
1262
1872
454
324
1132
138
1129
1175
391
844
290
1427
665
964
1206
885
725
180
1015
```

```
671
819
536
930
541
783
955
982
1262
1061
1540
445
609
639
485
1101
1149
432
2331
650
1461
1404
1226
1318
159
302
377
927
1143
664
346
1289
2208
1121
388
664
1586
1118
1012
408
337
1477
529
1457
323
284
104
628
482
920
1799
1159
2175
461
504
207
578
456
581
849
1468
```

```
906
1245
1346
1211
377
1990
709
383
196
1070
468
303
195
2320
913
617
1148
171
1293
388
1018
264
1258
460
477
1030
885
1388
685
646
1040
584
635
1297
323
1176
759
555
1188
1020
212
1354
360
502
1049
585
840
735
399
1277
623
712
1948
886
435
941
1112
1264
1194
242
1628
```

```
1162
744
1125
1264
706
1607
181
103
880
1378
1121
624
639
864
891
1732
996
646
1349
799
1821
855
559
1512
1389
887
442
1115
456
1941
407
986
1227
599
1443
1418
765
690
354
548
827
534
526
484
528
425
531
310
340
187
433
292
885
826
801
1308
996
1308
968
852
150
```

```
201
760
212
1519
1003
477
397
752
1436
1176
1340
979
481
1278
2163
232
1080
239
323
530
605
1427
1371
1429
754
1228
732
1020
1123
1080
75
577
262
247
486
700
432
1336
557
175
423
338
791
654
1522
344
384
835
548
331
381
764
646
777
560
1237
355
673
687
327
421
```

```
372
938
1365
898
420
755
449
1384
740
344
245
840
1556
1505
115
617
604
960
1564
1566
734
327
281
1261
154
645
690
450
845
174
1317
558
1395
1167
791
807
208
1267
435
1248
441
460
529
1774
470
801
213
431
653
136
701
1360
1336
1442
225
894
805
1512
975
429
1409
```

```
213
1497
522
531
553
1248
197
1249
743
887
364
542
825
1084
1028
387
292
858
1215
448
1169
565
969
878
1164
156
282
331
1676
860
505
1421
1209
342
817
1090
565
1175
1274
1213
1270
699
839
1367
2257
269
860
1385
666
899
733
646
640
2089
648
1178
663
519
134
509
1144
```

```
157
230
1114
553
783
1592
1211
1301
103
439
1167
1212
613
2302
258
1972
883
944
391
588
211
300
2047
1203
2212
822
667
714
1189
942
471
516
940
1129
1516
859
169
519
989
1291
925
87
614
306
791
942
835
1827
158
1037
921
1026
1080
1363
709
831
755
1269
262
213
1388
```

```
772
495
138
1285
1770
453
1363
381
231
1276
866
810
1393
287
927
379
1210
1067
398
643
1663
716
1680
```