```
import pandas as pd
from google.colab import drive
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import itertools
from random import uniform
import random
from sklearn.neighbors import KNeighborsClassifier
import copy
from math import ceil
```

loading the wine dataset:

```
dataset = pd.read_csv("drive/My Drive/Colab_Notebooks/wine.csv")
dataset.head()
```

```
   14.23  1.71  2.43  15.6  127   2.8  3.06  0.28  2.29  5.64  1.04  3.92  106
0  13.20  1.78  2.14  11.2  100  2.65  2.76  0.26  1.28  4.38  1.05  3.40  105
1  13.16  2.36  2.67  18.6  101  2.80  3.24  0.30  2.81  5.68  1.03  3.17  118
2  14.37  1.95  2.50  16.8  113  3.85  3.49  0.24  2.18  7.80  0.86  3.45  148
3  13.24  2.59  2.87  21.0  118  2.80  2.69  0.39  1.82  4.32  1.04  2.93   73
4  14.20  1.76  2.45  15.2  112  3.27  3.39  0.34  1.97  6.75  1.05  2.85  145
```

separating test and train and specifying total number of features

```
def load_dataset():
    dataset = pd.read_csv("drive/My Drive/Colab_Notebooks/wine.csv")

    x = dataset.iloc[:, 0:13].values
    y = dataset.iloc[:, 13].values

    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.4, shu
    feature_scaler = StandardScaler()
    X_train = feature_scaler.fit_transform(X_train)
    X_test = feature_scaler.transform(X_test)

    return X_train, X_test, y_train, y_test
```

```
def roulette_wheel_selection(l_sf):
    # the weights are obtained from choosing_sf function
    res = (random.choices(list(l_sf.keys()), weights=(
        0.13333333333333333, 0.125, 0.11764705882352941, 0.1111111111111111, 0.1
```

✓ 4s completed at 1:26 AM

```
        0.09090909090909091, 0.08695652173913043, 0.08333333333333333), k=1))
    return res[0]


def choosing_sf(f):
    l_sf = {}
    for sf in range(3, f):
        l = f - sf
        sum = 0
        for i in range(1, l + 1):
            sum += (f - i)
        lsf = (l / sum)
        l_sf[sf] = lsf
    return roulette_wheel_selection(l_sf)


def compute_similarity_by_correlation(X_train, f):
    df = pd.DataFrame(X_train)
    corr_matrix = df.corr(method='pearson')
    corr = {}
    for i in range(f):
        s = 0
        for j in range(f):
            if i != j:
                s += abs(corr_matrix[i][j])
        corr[i] = (s / (f - 1))
    sorted_corr = sorted(corr.items(), key=lambda x: x[1], reverse=True)
    similar = sorted_corr[:f // 2 + 1]
    dissimilar = sorted_corr[f // 2 + 1:]
    dissimilar.reverse()
    return dissimilar, similar


def initialize_population(f, size):
    population = []
    for _ in range(size):
        pos = [random.randint(0, 1) for _ in range(f)]
        population.append(pos)
    return population


def initialize_velocity(f, size):
    velocity = []
    for _ in range(size):
        vel = [random.uniform(0, 1) for _ in range(f)]
        velocity.append(vel)
    return velocity


def normalize(data, f):
    l = -1
    u = 1
    normalized_data = copy.deepcopy(data)
    normalized_data = copy.deepcopy(data)
    for i in range(f):
        x_max = np.max(data[:, [i]])
        x_min = np.min(data[:, [i]])
        tmp = (data[:, [i]] - x_min) / (x_max - x_min)
        normalized_column = l + ((u - l) * (tmp))
        normalized_data[:, [i]] = normalized_column
    return normalized_data


def move_particles(velocity, gbest, pbest, pos, size):
    c1 = c2 = 2
    v_max = 4
    v_min = -4
    for t in range(size):
        for i in range(len(pos[t])):
            velocity[t][i] += c1 * random.uniform(0, 1) * np.array(pbest[t][i] -
                              c2 * random.uniform(0, 1) * (gbest[i] - pos[t][i])
            if velocity[t][i] > v_max or velocity[t][i] < v_min:
                velocity[t][i] = max(min(velocity[t][i], v_max), v_min)

    s = copy.deepcopy(pos)
    for t in range(size):
        for i in range(len(pos[t])):
            s[t][i] += 1 / (1 + np.exp(-velocity[t][i]))
            pos[t][i] = 1 if random.uniform(0, 3) < s[t][i] else 0
    return pos, velocity


def remove(arr, similar, number_of_similars, ns):
    diff = number_of_similars - ns
    while diff > 0:
        for i in similar:
            if i[0] in arr:
                arr.remove(i[0])
                diff -= 1
                break
    return arr


def add(arr, dissimilar, number_of_dissimilars, nd):
    diff = nd - number_of_dissimilars
    while diff > 0:
        for i in dissimilar:
            if i[0] not in arr:
                arr.append(i[0])
                diff -= 1
                break
    return arr


def feature_selection(pos, similar, dissimilar, ns, nd):
    number_of_similars = 0
    for i in similar:
        for j in pos:
```

```
        for j in pos:
            if i[0] == j:
                number_of_similars += 1
    if number_of_similars > ns:
        pos = remove(pos, similar, number_of_similars, ns)

    number_of_dissimilars = 0
    for i in dissimilar:
        for j in pos:
            if i[0] == j:
                number_of_dissimilars += 1
    if number_of_dissimilars < nd:
        pos = add(pos, dissimilar, number_of_dissimilars, nd)

    return pos


def fitness(X_train, X_test, y_train, y_test, pos, f, similar, dissimilar, gbest
    alpha = 0.65
    ns = ceil(sf * alpha)
    nd = ceil(sf * (1 - alpha))

    X_train_normalized = normalize(X_train, f)
    X_test_normalized = normalize(X_test, f)

    count = 0
    for p in pos:
        x_t = copy.deepcopy(X_train_normalized)
        arr = [j for j, i in enumerate(p) if i == 1]
        selected_features = feature_selection(arr, similar, dissimilar, ns, nd)
        x_t = x_t[:, selected_features]
        if len(selected_features) == 13:
            continue
        neigh = KNeighborsClassifier(n_neighbors=3)
        prediction = neigh.fit(x_t, y_train)
        x_test = copy.deepcopy(X_test_normalized)
        x_test = x_test[:, selected_features]
        res = neigh.score(x_t, y_train, sample_weight=None)
        if res > gbest_fit:
            gbest_fit = res
            gbest_pos = p
        if res > pbest_fit[count]:
            pbest_fit[count] = res
            pbest_pos[count] = p
        count += 1
    print('score of train dataset:', gbest_fit)
    return gbest_pos, gbest_fit, pbest_pos, pbest_fit



X_train, X_test, y_train, y_test = load_dataset()
f = X_train.shape[1]
population_size = 20
sf = choosing_sf(f)
dissimilar, similar = compute_similarity_by_correlation(X_train, f)
```

```
                              _          _  _             _

pos = initialize_population(f, population_size)
vel = initialize_velocity(f, population_size)
gbest_pos = pos[0]
gbest_fit = 0
pbest_pos = pos
pbest_fit = [0] * population_size

gbest_pos, gbest_fit, pbest_pos, pbest_fit = fitness(X_train, X_test, y_train, y
                                                      gbest_pos, gbest_fit, pbes
for _ in range(50):
    pos, vel = move_particles(vel, gbest_pos, pbest_pos, pos, population_size)
    gbest_pos, gbest_fit, pbest_pos, pbest_fit = fitness(X_train, X_test, y_trai
                                                          dissimilar, gbest_pos,
neigh = KNeighborsClassifier(n_neighbors=3)
x_test = X_test[:, gbest_pos]
x_test = normalize(x_test, f)
prediction = neigh.fit(x_test, y_test)
res = neigh.score(x_test, y_test, sample_weight=None)
print('score of the test dataset: ', res)

# the final score on test dataset is 90-+ 3%
```

```
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of train dataset: 0.9905660377358491
score of the test dataset:  0.8873239436619719
```