
西安電子科技大學
XIDIAN UNIVERSITY

# 模式识别大作业四

## 特征选择的遗传算法（GA）

任俊杰

1702052

17170120015

# 模式识别 特征选择的遗传算法

## 一、特征选择的遗传算法介绍

### 1.1 特征选择

特征选择在概念上十分简单，即对原有特征进行删选优化。特征选择就是从原始特征中挑选出一些最有代表性、分类性能最好的特征进行分类。

特征选择要解决两个问题：如何确定选择的标准，如可分离性判据；如何构建快速寻优算法。

特征选择根据是否直接考虑分类器性能可以分为两种方法：

Filter 方法：根据独立于分类器的指标 J 来评价所选择的特征子集 S，在所有可能的特征子集中搜索出使得 J 最大的特征子集作为最优特征子集。 不考虑所使用的学习算法。

Wrapper 方法：将特征选择和分类器结合在一起，在分类过程中表现优异的特征子集会被选中。

### 1.2 遗传算法（Genetic Algorithm）

#### （1）算法原理

遗传算法的思路来自人们对生物进化过程的认识。关于生物进化，人们目前比较普遍接受的认识是，每种生物的染色体都在以一定的概率发生各种突变和重组，这些突变和重组是随机发生的，它们统称为变异。生物生存的环境对变异的结果有选择的作用，如果一种变异导致生物能更好地适应环境，则这种变异就被保留下来传给后代；而如果一种变异导致生物对环境的适应性变差，这种变异就会逐渐消失。

这一思想在 20 世纪 60 年代被当时正在读博士研究生的 JohnH.Holland 借鉴到优化问题中，他把优化问题比喻为在无数可能的重组和突变组合中发现适应性最强的组合的问题，设计了种特殊的搜索算法，模拟自然界中通过有性繁殖迅速增加群体多样性以更快地出现适应性强的组合的过程。他的专著在 1975 年出版，之后多次再版。遗传算法很快成为最有影响的优化算法之一，并且开创了一个新的领域。

遗传算法把候选对象编码为一条染色体 (chromesome)，在特征选择中，如果目标是从 D 个特征中选择 d 个，则把所有特征描述为一条由 D 个 0/1 字符组成的字符串，0 代表该特征没有被选中，1 代表该特征被选中，这个字符串就叫做染色体，记作 m。显然，要求的是一条有且仅有 d 个 1 的染色体，这样的染色体共有 $C_D^d$ 种。 优化的目标被描述成适应度（fitness）函数，每一条染色体对应一个适应度值 $f(m)$ 。可以用前面定义的基于类内类间距离的类别可分性判据作为适应度。

**基因链码：**待解问题的解的编码，每个基因链码也称为一个个体。对于特征选择，可用一个 D 位的 0/1 构 成的串表示一种特征组合。

**群体：**若干个个体的集合，即问题的一些解的集合。

**交叉：**由当前两个个体的链码交叉产生新一代的两个个体。

**变异：**由一个链码随机选取某基因使其翻转

**适应度：**每个个体 $x_i$ 的函数值 $f_i$ ,个体$x_i$ 越好，适应度 $f_i$越大。新 一代群体对环境的平均适应度比父代高

**（2）遗传算法流程**

① 令进化代数 t=0。

② 给出初始化群体 P(t)，令$x_g$为任一个体。

③ 对 P(t)中每个个体估值，并将群体中最优解 x′ 与$x_g$比较， 如果 x′的性能优于$x_g$，则$x_g$=x′ 。

④ 如果终止条件满足，则算法结束，$x_g$为算法的结果。否则继续。

⑤ 从 P(t)中选择个体并进行交叉和变异操作,得到新一代 群体 P(t+1)。令 t=t+1,转到 ③

**（3）算法优缺点**

**1、优点:**

1) 与问题领域无关切快速随机的搜索能力。
2) 搜索使用评价函数启发，过程简单
3) 使用概率机制进行迭代，具有随机性。
4) 具有可扩展性，容易与其他算法结合。
5) 遗传算法具有良好的全局搜索能力，可以快速地将解空间中的全体解搜索出，而不会陷入局部最优解的快速下降陷阱；是全局优化算法，一般的迭代方法容易陷入局部极小的陷阱而出现"死循环"现象，使迭代无法进行。遗传算法很好地克服了这个缺点，是一种全局优化算法。

**2、缺点:**

1) 遗传算法的编程实现比较复杂,首先需要对问题进行编码,找到最优解之后还需要对问题进行解码,
2) 另外三个算子的实现也有许多参数,如交叉率和变异率,并且这些参数的选择严重影响解的品质,而目前这些参数的选择大部分是依靠经验.

3) 没有能够及时利用网络的反馈信息,故算法的搜索速度比较慢，要得要较精确的解需要较多的训练时间。

4) 算法对初始种群的选择有一定的依赖性，能够结合一些启发算法进行改进。

## 二、实验数据集介绍

### 2.1 Iris 数据集介绍

| Iris 数据集 | | | |
|---|---|---|---|
| 类别 | 3 | | |
| 维度 | 4 | | |
| 数据长度 | 150 | 50 | Iris-setosa |
| | | 50 | Iris-versicolor |
| | | 50 | Iris-setosa |

### 2.2 Sonar 数据集介绍

| Sonar 数据集 | | | |
|---|---|---|---|
| 类别 | 2 | | |
| 维度 | 60 | | |
| 数据长度 | 208 | 97 | R |
| | | 111 | M |

# 三、实验设置

## 3.1 算法流程

本次实验选用的**是 K 近邻法**作为分类器，来评价所选特征的好坏，用来计算种群的适应度函数。由于上述算法流程较为简略，在实际编程实现对编码、选择、交叉、变异又使用了一些其他方法，下面具体说明整体流程（由于 iris 数据集维度较低，故使用 sonar 数据集具体说明）：

**1、读取数据集，按照一定比例随机分为测试集、训练集**

**2、编码及种群的初始化：**

对所有特征描述为一个由 60 个 0/1 元素组成的列表，代表染色体，0 代表该特征没有被选中，1 代表该特征被选中。

用户给定选择的数据维度 d 和种群数目 p，将染色体（列表）初始化全 0，然后随机将染色体中的 d 位变为 1，这样就从 60 维特征中随机选择了 d 维。重复上述过程 p 次，则可以得到一个有 p 条染色体的初代种群 P(0)，并且每条染色体都不尽相同。并随机选定一个最优个体。

**3、开始迭代，直至达到指定的迭代次数：**

**1） 更新每个个体的适应度函数**，并更新最优个体、统计种群的平均适应度。计算每个个体的适应度时，根据个体的染色体情况，将没有选中的特征删除（包括训练集和测试集），将数据送入分类器，计算分类准确率作为该个体的适应度函数。分类器使用之前实现的 K 近邻，根据之前的经验，K 取 2。

**2） 根据适应度函数，对种群进行轮盘赌选择。**对种群中的染色体进行采样，由采样出的染色体经过一定的操作繁殖出下一代染色体，组成下一代的种群。这里实用的是轮盘赌的方式进行选择：首先将种群中所有个体的适应

度归一化，$f_{i_{归一}} = \frac{f_i}{\sum_i^p f_i}$，将这些适应度值累加起来，得到 p 个区间，每个区间的长度代表对应染色体的适应度值。从 0-1 之间取随机数，该数落到哪个区间，就选用哪条染色体。为保证种群的染色体数目不变，重复 p 次，就得到了基于上一代种群适应度的新子代种群。

**3）交叉。**对一条染色体，按照一定的概率（交叉概率设为 0.4），选择是否进行交叉操作。再从种群中随机寻找到另外一条父代染色体，与之进行交叉，为了使交叉之后的染色体特征维数不变，采用了如下方法：

先从一个父代染色体中随机选取一个片段（长度为 60/2=30），统计该片段中有几个为 1 的基因，再从与之匹配的父代染色体中寻找长度相同、1 基因数目相同的片段，若找到，则进行交叉操作；若未找到，则寻找下一对父代染色体，直到将所有父代种群遍历完毕。

**4）变异。**同样的，对一条染色体，按照一定的概率（变异概率设为 0.01），选择是否进行变异操作。为了使变异之后染色体中特征数量不变，又采用了以下方法：

随机从种群中选择一个个体，再随机地选择一个基因进行反转，若该基因由 1 变为了 0，则再随机选一个 0 变成 1，反之也执行同样的操作。直至遍历完种群中所有的个体。

**5）重复迭代。**在进行完选择、交叉、和变异操作之后，上一代的种群已经变成了新一代的种群。重复上述过程 1-4，在遗传算法迭代的过程中，种群中的染色体会趋于所选特征数中的最优解，达到一定的迭代次数 t 后，算法停止，输出最终种群中适应度值最大的染色体，和每次迭代过程中的种群平均适应度，即完成了在 D 维特征中选择 d 个最优的特征。

### 3.2 在数据集上验证遗传算法

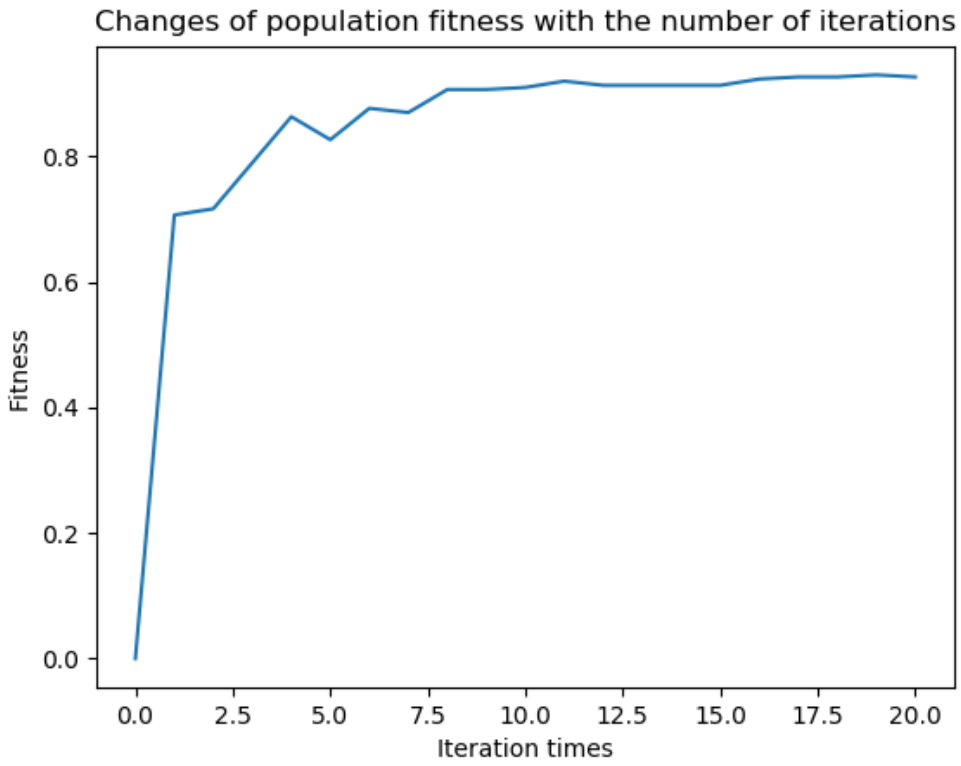
1、首先**取指定维度**特征（iris2 维、sonar30 维）的时候，绘图展示每一次迭代中种群的适应度，观察随着遗传算法的迭代，每一代种群的适应度变化情况。

2、下面扩展到**取不同维度**特征的情况。iris 取 1-4 维、sonar 取 1-60 维，绘图展示随维度增加，种群中最优个体的适应度变化情况。

## 四、实验结果展示与分析

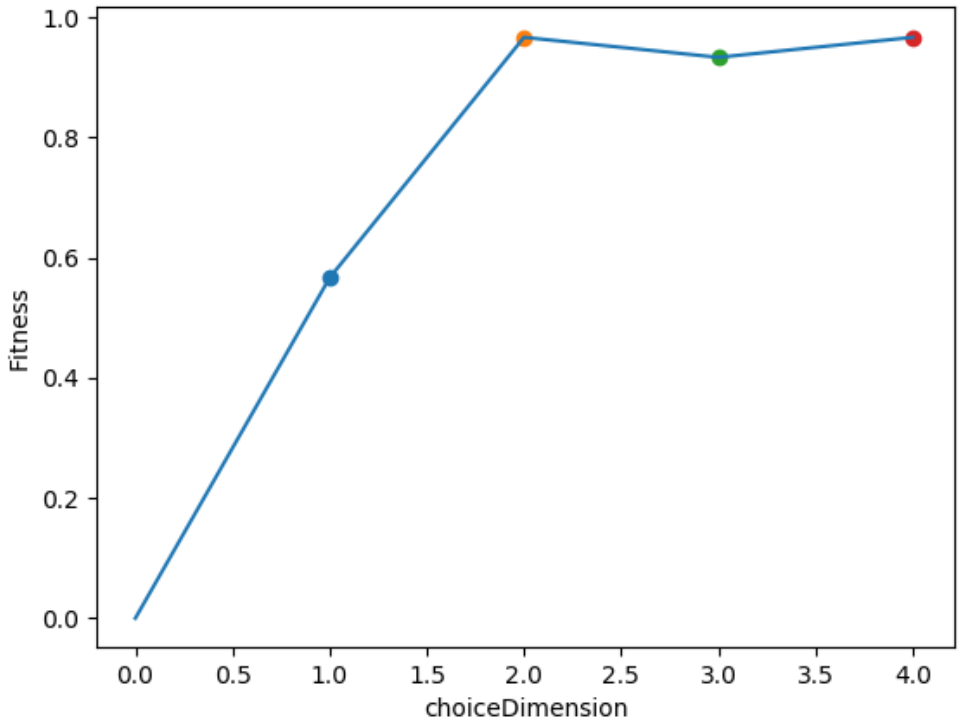
### 4.1 在 iris 数据集上验证遗传算法

**指定维度：**因为 iris 数据集只有 4 维可供选择，所有，我选择了 2 维情况下验证遗传算法的收敛情况，种群规模为 10，迭代 20 次，每次迭代的种群平均适应度如下图所示：



从图中可以看到，随着迭代次数的增加，种群的适应度函数逐渐提高，迭代 20 次后已经基本收敛，故不再继续迭代。但在多次试验中，也出现了适应度忽高忽低的情况（在下面 sonar 数据集上尤为明显，iris 数据集上出现较少，不做展示)，经分析，可能是出现了不适应环境的交叉、变异，但由于使用的是轮盘赌选择的原因，该染色体还是有一定可能性被保留，若该染色体多次被选中，将会产生较大影响

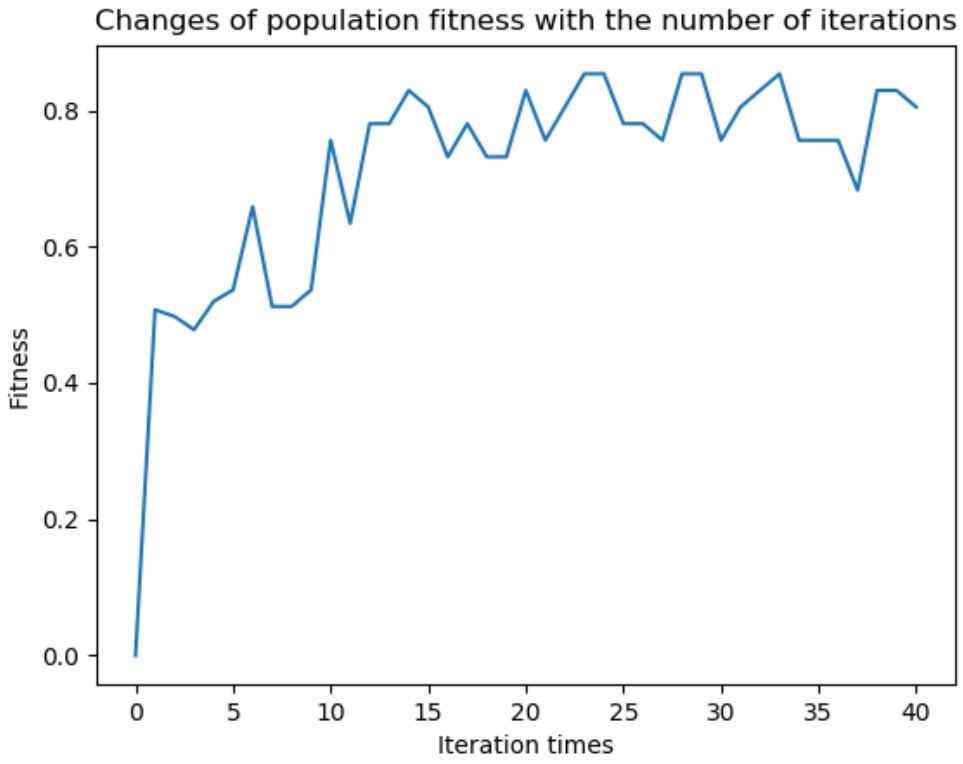
**不同维度：**在 iris 数据集上，分别取维度 1-4，种群规模设置为 5，迭代 10 次，选取不同维度时种群经遗传算法特征选择后的种群最优个体的适应度如下图所示：




从上图可以看出，选择 1 维时，情况最差，在 2-4 维的情况下，最终适应度差别不大，可见；iris 数据集选择两维就可以达到不错的分类效果（K 近邻)。

### 4.2 在 sonar 数据集上验证遗传算法

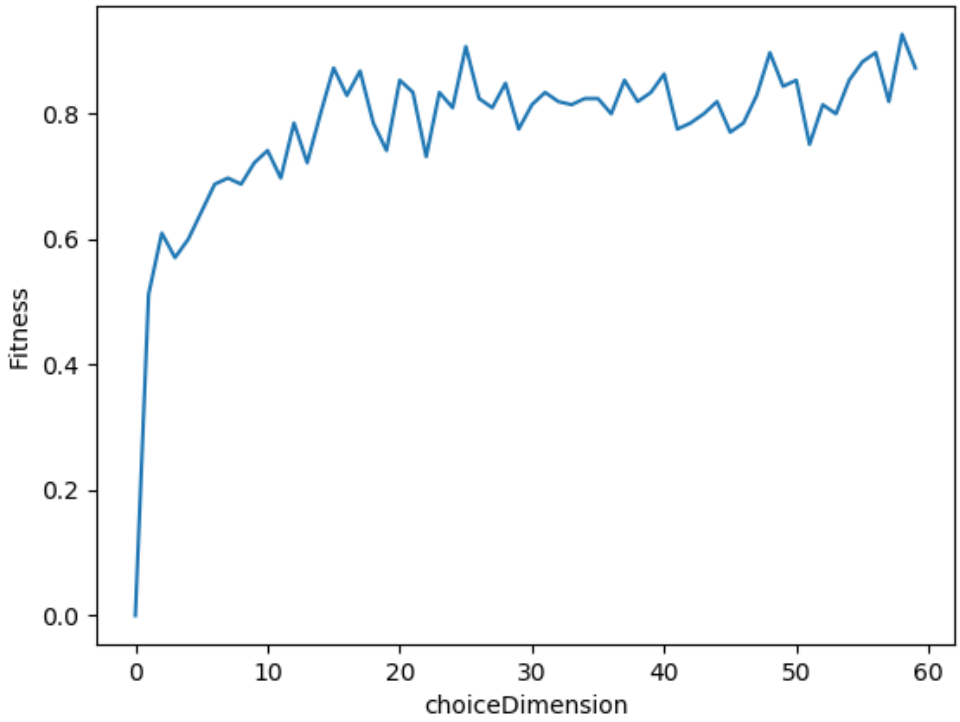
**指定维度：**因为 sonar 数据集有 60 维可供选择，所有，我选择了 30 维情况下验证遗传算法的收敛情况，种群规模为 10，迭代 40 次，每次迭代的种群平均适应度如下图所示：




从图中可以看到，随着迭代次数的增加，适应度波动较大，但种群的适应度函数也逐渐提高，迭代 15 次后已经基本稳定在 0.75 附近。由于所选择交叉、变异、轮盘赌选择的方法原因，在维度比较大的时候，不可能保证无限次迭代下去种群的平均适应度会归一到一个具体数值，每次都会有新的染色体出现，务必会影响种群的平均适应度，故出现波动较大的情况，但整体走势符合正常增大的趋势。

**不同维度：**在 sonar 数据集上，分别取维度 1-60，种群规模设置为 5，迭代 20 次，选取不同维度时种群经遗传算法特征选择后的种群最优个体的

适应度如下图所示：




从上图可以看出，选择维度在 15 维之前，随选择维度提升，种群适应度也在提升，20 维之后，适应度基本稳定到 0.8 左右波动，不再有明显的上升趋势。

### 4.3 总结

本次遗传算法的实现，由于所有代码 100%均是自己所写，并没有使用任何现成的工具包(包括数据读取和基本的选择、遗传、变异)，只用了 random、numpy、matplotlib 等基础包。评价指标使用的 K 近邻也是根据上上次自己实现的代码所改。加上自己对算法各个细节的调优不足，所以最终算法的效果波动较大，只有大致的变化趋势，和别人封装好的包相比，差距还是较为明显，深感自己不足。但由于时间和能力原因，有些小 bug 还没有优化，但是已经定位 (比如说：自交后染色体莫名混乱，已经确定是哪个函数的问题，但耗了很长时间仍未解决)，之后如果有机会多加改进，谢谢。

# 五、Python 代码

## 5.1 iris 数据集（由于使用 sonar 数据集的代码差别很小，只需要改分类的名称和算法调用的一行代码即可，故不再贴出代码）

```
import os
import sys
import numpy as np
from numpy import *
import random
import matplotlib.pyplot as plt

def readData(DATA_PATH,DATA_SIZE,DATA_DIMENSION,test_rate):
    test_Labels = random.sample(range(0,DATA_SIZE-
1),int(DATA_SIZE*test_rate))
    test_Data = [[0 for i in range(DATA_DIMENSION+1)] for j in range(le
n(test_Labels))]
    train_Data = [[0 for i in range(DATA_DIMENSION+1)] for j in range(D
ATA_SIZE-len(test_Labels))]
    #随机提取训练、测试数据
    f = open(DATA_PATH)
    line = f.readline()
    i = 0
    train_Label = 0
    test_Label = 0
    while line:
        temp = line.split(",",DATA_DIMENSION)
        if i in test_Labels:
            test_Data[test_Label][0:DATA_DIMENSION] = list(map(float, t
emp[0:DATA_DIMENSION]))
            test_Data[test_Label][DATA_DIMENSION] = temp[DATA_DIMENSION
]
            test_Label = test_Label+1
        else:
            train_Data[train_Label][0:DATA_DIMENSION] = list(map(float,
 temp[0:DATA_DIMENSION]))
            train_Data[train_Label][DATA_DIMENSION] = temp[DATA_DIMENSI
ON]
            train_Label = train_Label+1
        i = i+1
        line = f.readline()
    #将要训练的数据分类保存
    _1_Num = _2_Num = _3_Num = 0
```

```
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'Iris-setosa\n':
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-versicolor\n':
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Num = _3_Num+1
    _1_Data = np.mat(np.zeros((_1_Num,DATA_DIMENSION)))
    _2_Data = np.mat(np.zeros((_2_Num,DATA_DIMENSION)))
    _3_Data = np.mat(np.zeros((_3_Num,DATA_DIMENSION)))
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'Iris-setosa\n':
            _1_Data[_1_Num] = train_Data[i][0:DATA_DIMENSION]
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-versicolor\n':
            _2_Data[_2_Num] = train_Data[i][0:DATA_DIMENSION]
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Data[_3_Num] = train_Data[i][0:DATA_DIMENSION]
            _3_Num = _3_Num+1
    f.close()
    return train_Data,test_Data,_1_Data.T,_2_Data.T,_3_Data.T

#********************************************************* KNN **********
*******************************************
def dis_Sort(allDisInfo):
    for i in range(0,len(allDisInfo)):
        for j in range(0,len(allDisInfo)-i-1):
            if allDisInfo[j][1]>allDisInfo[j+1][1]:
                temp = allDisInfo[j]
                allDisInfo[j] = allDisInfo[j+1]
                allDisInfo[j+1] = temp
    return allDisInfo

#计算欧氏距离
def calcu_Eucli_Dis(x,y):
    dis = np.sqrt(np.sum(np.square(np.array(x)-np.array(y))))
    return dis

def k_Judge(allDisInfo,k):
    type_1 = type_2 = type_3 = 0
    for i in range(k):
        if allDisInfo[i][0] == "Iris-setosa\n":
```

```
            type_1 += 1
        if allDisInfo[i][0] == "Iris-versicolor\n":
            type_2 += 1
        if allDisInfo[i][0] == "Iris-virginica\n":
            type_3 += 1
    if type_1>=type_2 and type_1>=type_3:
        return "Iris-setosa\n"
    if type_2>=type_1 and type_2>=type_3:
        return "Iris-versicolor\n"
    if type_3>=type_1 and type_3>=type_2:
        return "Iris-virginica\n"

def K_Nearest_Neighbors(train_Data,test_Data,dataDimension,k=5):
    #数据准备
    # print("**********************************************************
**********")
    # print("                    train_Size:",len(train_Data))
    # print("                    test_Size:",len(test_Data))
    # print("**********************************************************
**********")

    wrongNum = correctNum = 0
    #遍历每一个测试元素
    for testElement in range(len(test_Data)):
        allDisInfo = [[0 for i in range(2)] for j in range(len(train_Da
ta))]  #[forcast_label][distance]

        # print("**********************testElement_",testElement,"*****
******************")
        #每一个测试元素与训练集中的每一个数据进行距离计算
        for trainElement in range(len(train_Data)):
            testArray = np.array(test_Data[testElement][0:dataDimension
],float)
            trainArray = np.array(train_Data[trainElement][0:dataDimens
ion],float)
            #计算距离
            dis = calcu_Eucli_Dis(testArray,trainArray)
            #保存所有 label 的距离信息
            allDisInfo[trainElement][0] = train_Data[trainElement][data
Dimension]
            allDisInfo[trainElement][1] = dis

        #对距离信息进行排序
        dis_Sort(allDisInfo)
```

```
        # for i in range(len(allDisInfo)):
        #     print(allDisInfo[i])
        #选择 K 值进行判别
        forcast_label = k_Judge(allDisInfo,k)
        # print("----------------------------------
-> Forcast Label:",forcast_label,"----------------------------------
-> Real Label:",test_Data[testElement][dataDimension])
        if(test_Data[testElement][dataDimension] == forcast_label):
            correctNum += 1
        else:
            wrongNum += 1

    # print("-----------------------------------")
    # print("|WrongNums:",wrongNum,"correctNums:",correctNum)
    # print("-----------------------------------\n\n\n")
    Accuracy = correctNum/(correctNum+wrongNum)
    return  Accuracy

#********************************************************** Genetic
Algorithm ************************************************************
*******
# 每个个体的类
class individual:
    def __init__(self,dataDimension,choiceDimension):
        super().__init__()
        init_chrom_val = random.sample(range(0,dataDimension),choiceDim
ension)
        print(init_chrom_val)
        self.dataDimension = dataDimension  # 原始数据维度
        self.fitness = 0                    # 适应度
        self.trainArray = []
        self.testArray = []
        self.Chromosome = [0 for i in range(dataDimension)]        # 初
始化染色体,随机选中 d 个特征
        for i in range(choiceDimension):
            self.Chromosome[init_chrom_val[i]] = 1

    def caclu_fitness(self,trainData,testData):
        deletePos = []
        for i in range(len(self.Chromosome)):
            if not self.Chromosome[i]:
                deletePos.append(i)
```

```
        self.trainArray = np.delete(np.array(trainData),deletePos,1).to
list()
        self.testArray = np.delete(np.array(testData),deletePos,1).toli
st()

        self.fitness = K_Nearest_Neighbors(self.trainArray,self.testArr
ay,self.dataDimension-len(deletePos))

def seletction(population): # 使用轮盘赌进行个体选择
    next_population = []

    # 归一化所有个体的适应度函数，以进行轮盘赌选择
    fitness_sum = 0
    fitness_norm = [0 for i in range(len(population))]
    for i in range(len(population)):
        fitness_sum += population[i].fitness

    start_pos = 0
    for i in range(len(population)):
        fitness_norm[i] = population[i].fitness/fitness_sum + start_pos
        start_pos = fitness_norm[i]

    for indiv in range(len(population)):    # 选出新一代种群，种群规模保持
不变
        rand_num = random.random()
        for j in range(len(population)):
            if rand_num <= fitness_norm[j]:
                break

        print("individual_",j,"_:is selected")
        next_population.append(population[j])
    return next_population


# 交叉
def cross(population,cross_prob = 0.6):
    for i in range(len(population)):
        if random.random() <= cross_prob:
            # 一次重组 D//2 个基因，并且要保证交叉的染色体序列长度相同，含选中
特征的数量相同

            # 统计交叉染色体中 1 的数量
            gen_1_num = 0
            for gen in range(len(population[i].Chromosome)//2):
```

```
                if population[i].Chromosome[gen] == 1:
                    gen_1_num += 1

            # 随机寻找另一个个体，并寻找到相匹配的染色体片段
            another_indiv = random.randint(0,len(population)-1) # 与之交
配的个体
            another_cross_pos = 0
            while 1:
                # print(i,another_indiv,another_cross_pos)
                another_gen_1_num = 0
                for gen in range(another_cross_pos,len(population[i].Ch
romosome)//2+another_cross_pos):
                    if population[another_indiv].Chromosome[gen] == 1:
                        another_gen_1_num += 1

                if another_gen_1_num == gen_1_num:
                    print(i,another_indiv,"pos",another_cross_pos)
                    tmp = population[i].Chromosome[0:len(population[i].
Chromosome)//2]
                    population[i].Chromosome[0:len(population[i].Chromo
some)//2] = population[another_indiv].Chromosome[another_cross_pos:len(
population[i].Chromosome)//2+another_cross_pos]
                    population[another_indiv].Chromosome[another_cross_
pos:len(population[i].Chromosome)//2+another_cross_pos] = tmp
                    break
                else:
                    another_cross_pos += 1

                if len(population[i].Chromosome)//2+another_cross_pos >
 len(population[i].Chromosome):  #超出范围，寻找失败
                    print(i,"fail to find another position")
                    break
# 存在自交问题
def cross_v2(population,cross_prob = 0.4):
    for i in range(len(population)//2):
        if random.random() <= cross_prob:
            # 一次重组 D//2 个基因，并且要保证交叉的染色体序列长度相同，含选中
特征的数量相同

            # 统计交叉染色体中 1 的数量
            cross_pos = random.randint(0,len(population[i].Chromosome)/
/2-1)
            cross_pos_stop = cross_pos+len(population[i].Chromosome)//2
            gen_1_num = 0
```

```
            for gen in range(cross_pos,cross_pos_stop):
                if population[i].Chromosome[gen] == 1:
                    gen_1_num += 1

            # 随机寻找另一个个体，并寻找到相匹配的染色体片段
            another_indiv = i
            while i == another_indiv:
                another_indiv = random.randint(0,len(population)//2-
1) # 与之交配的个体

            another_cross_pos = 0
            another_cross_pos_stop = len(population[i].Chromosome)//2+a
nother_cross_pos
            # print("stop",another_cross_pos_stop)
            while 1:
                another_gen_1_num = 0
                for gen in range(another_cross_pos,another_cross_pos_st
op):
                    if population[another_indiv].Chromosome[gen] == 1:
                        another_gen_1_num += 1
                if another_gen_1_num == gen_1_num:
                    print("individual_",i,"_and individual_",another_in
div,"_ have crossed at positon: 0 and position:",another_cross_pos)
                    # print("yuan:",population[i].Chromosome[cross_pos:
cross_pos_stop],len(population[i].Chromosome[cross_pos:cross_pos_stop])
)
                    # print("huan:",population[another_indiv].Chromosom
e[another_cross_pos:another_cross_pos_stop],len(population[another_indi
v].Chromosome[another_cross_pos:another_cross_pos_stop]))
                    tmp = population[i].Chromosome[cross_pos:cross_pos_
stop]
                    population[i].Chromosome[cross_pos:cross_pos_stop]
= population[another_indiv].Chromosome[another_cross_pos:another_cross_
pos_stop]
                    population[another_indiv].Chromosome[another_cross_
pos:another_cross_pos_stop] = tmp
                    break
                else:
                    another_cross_pos += 1
                    another_cross_pos_stop += 1

                if another_cross_pos_stop >= len(population[i].Chromoso
me):  #超出范围，寻找失败
                    print(i,"fail to find another position")
```

```
                    break
# 变异
def variation(population,variation_prob = 0.01):
    for i in range(len(population)):
        if random.random() <= variation_prob:
            variation_pos = random.randint(0,len(population[i].Chromoso
me)-1)       # 变异的位置
            another_pos = random.randint(0,len(population[i].Chromosome
)-1)
            # 变异后要保证选中的特征数量不变
            if (population[i].Chromosome[variation_pos] == 1):  #变异位
置原来为1就找到位置为0的也变异
                while population[i].Chromosome[another_pos] != 0:
                    another_pos = random.randint(0,len(population[i].Ch
romosome)-1)
                population[i].Chromosome[another_pos] = 1
                population[i].Chromosome[variation_pos] = 0
            else:                                                #变异位
置原来为0就找到位置为1的也变异
                while population[i].Chromosome[another_pos] != 1:
                    another_pos = random.randint(0,len(population[i].Ch
romosome)-1)
                population[i].Chromosome[another_pos] = 0
                population[i].Chromosome[variation_pos] = 1
            print("individual_",i,"_: have variated,position:[",variati
on_pos,another_pos,"]")


def GA(dataPath,dataSize,dataDimension,choiceDimension,populationSize,S
TOP_TIMES):
    # 加载数据集
    trainData,testData,_,_,_ = readData(dataPath,dataSize,dataDimension
,1/5)
    # 初始化种群 和 最优个体
    population = [individual(dataDimension,choiceDimension) for i in ra
nge(populationSize)]

    fit_ave = [0]
    interation_times = 0    #迭代次数
    # 开始迭代:
    while interation_times < STOP_TIMES:
        great_indiv = population[0]
        fitness_ave = 0
        interation_times += 1
```

```
        print("********************** _generation:",interation_times,
"_**************************")
        # 更新每个个体的适应度函数,并更新最优个体
        for i in range(populationSize):
            population[i].caclu_fitness(trainData,testData)
            fitness_ave += population[i].fitness

            if population[i].fitness > great_indiv.fitness:
                great_indiv = population[i]
            print("individual_",i,"_:",population[i].Chromosome,populat
ion[i].fitness)
        print("best:",great_indiv.Chromosome,great_indiv.fitness)

        fitness_ave /= populationSize
        fit_ave.append(fitness_ave)

        # 轮盘赌选择
        population = seletction(population)

        # 交叉
        cross(population, cross_prob = 0.4)

        # 变异
        variation(population, variation_prob = 0.01)

    return great_indiv,fit_ave

# 同一维度看收敛情况
if __name__ == "__main__":
    great_indiv,fit_ave = GA("iris.data",150,4,choiceDimension=2,popula
tionSize=10,STOP_TIMES=20)
    for i in range(len(fit_ave)):
        print("Fitness",i,": ",fit_ave[i])
        plt.scatter(i,fit_ave[i])
    plt.title("Changes of population fitness with the number of iterati
ons")
    plt.xlabel("Iteration times")
    plt.ylabel("Fitness")
    # plt.ylim(0,2)
    plt.plot(fit_ave)
    plt.show()

# 不同维度
# if __name__ == '__main__':
```

```
#     Accuracy_AVE = 0
#     Accuracy = []
#     Accuracy.append(0)
#     for dimen in range(1,5):
#         great_indiv,_ = GA("iris.data",150,4,choiceDimension=dimen,po
pulationSize=5,STOP_TIMES=10)
#         Accuracy.append(great_indiv.fitness)
#         plt.scatter(dimen,great_indiv.fitness)
#         Accuracy_AVE += great_indiv.fitness
#     Accuracy_AVE /= 4

#     for i in range(5):
#         print("Accuracy_",i+1,": ",Accuracy[i])
#     print("Accuracy_AVE:",Accuracy_AVE)
#     plt.title("Accuracy")
#     plt.xlabel("choiceDimension")
#     plt.ylabel("Fitness")
#     # plt.ylim(0,2)
#     plt.plot(Accuracy)

    # plt.show()
```