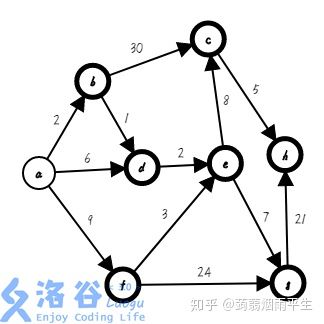
## 1.最短路径

有向图到各点的最短距离：

类似：这种图或者邻接矩阵的形式，可以得到某一点a向各个节点的最短路径



算法描述：

1. 初始化，以该节点更新与有向连接的邻结点的距离进行更新，不能通过的，两者之间距离为正无穷、
2. 找出最短路的结点b，确定该路径ab是a到b的最短路，并将b作为下一节点，继续重复算法一直到结束。
3. 以该节点更新与有向连接的邻结点的距离进行更新，
4. 重复23过程一直到结束

#include<fstream>  
#define MaxNum 765432100  
using namespace std;  
  
ifstream fin("Dijkstra.in");  
ofstream fout("Dijkstra.out");  
  
int Map[501][501];  
bool is\_arrived[501];  
int Dist[501],From[501],Stack[501];  
int p,q,k,Path,Source,Vertex,Temp,SetCard;  
  
int FindMin()  
{  
int p,Temp=0,Minm=MaxNum;  
for(p=1;p<=Vertex;p++)  
if ((Dist[p]<Minm)&&(!is\_arrived[p]))  
{  
Minm=Dist[p];  
Temp=p;  
}  
return Temp;  
}  
int main()  
{  
memset(is\_arrived,0,sizeof(is\_arrived));  
  
fin >> Source >> Vertex;  
for(p=1;p<=Vertex;p++)  
for(q=1;q<=Vertex;q++)  
{  
fin >> Map[p][q];  
if (Map[p][q]==0) Map[p][q]=MaxNum;  
}  
for(p=1;p<=Vertex;p++)  
{  
Dist[p]=Map[Source][p];  
if (Dist[p]!=MaxNum)   
From[p]=Source;  
else   
From[p]=p;  
}  
  
is\_arrived[Source]=true;  
SetCard=1;  
do  
{  
Temp=FindMin();  
if (Temp!=0)  
{  
SetCard=SetCard+1;  
is\_arrived[Temp]=true;  
for(p=1;p<=Vertex;p++)  
if ((Dist[p]>Dist[Temp]+Map[Temp][p])&&(!is\_arrived[p]))  
{  
Dist[p]=Dist[Temp]+Map[Temp][p];  
From[p]=Temp;  
}  
}  
else  
break;  
}  
while (SetCard!=Vertex);  
  
for(p=1;p<=Vertex;p++)  
if(p!=Source)  
{  
fout << "========================\n";  
fout << "Source:" << Source << "\nTarget:" << p << '\n';  
if (Dist[p]==MaxNum)  
{  
fout << "Distance:" << "Infinity\n";  
fout << "Path:No Way!";  
}  
else  
{   
fout << "Distance:" << Dist[p] << '\n';  
k=1;  
Path=p;  
while (From[Path]!=Path)  
{  
Stack[k]=Path;  
Path=From[Path];  
k=k+1;  
}  
fout << "Path:" << Source;  
for(q=k-1;q>=1;q--)  
fout << "-->" << Stack[q];  
}  
fout << "\n========================\n\n";  
}  
  
fin.close();  
fout.close();  
return 0;  
}

## 💯 2.数值概率算法和舍伍德算法。

## 3.灰色预测

这是像线性回归一下，建立起一个预测模型，然后进行预测未知数据

什么情况下使用灰色预测模型？

**数据量小，预测的周期短，预测的长度短**

分为GM(1,1)，GM(1,m)，GM(n,m)分别用于一个自变量一个因变量，多个自变量一个因变量，多个自变量多个因变量

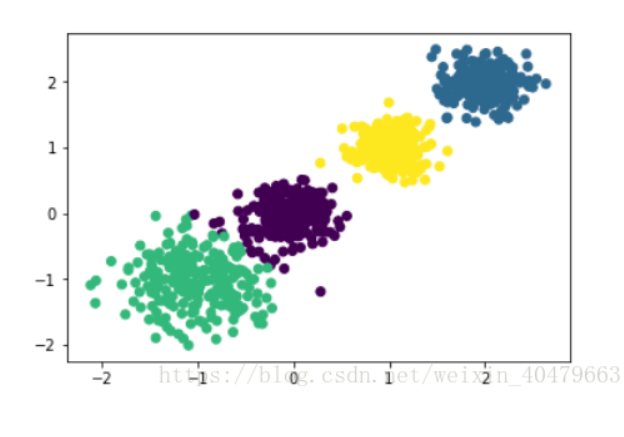
这里有两个程序，有一个简单版本，有一个复杂版本，复杂版本请参考github

function []=greymodel(y)  
% 本程序主要用来计算根据灰色理论建立的模型的预测值。  
% 应用的数学模型是 GM(1,1)。  
% 原始数据的处理方法是一次累加法。  
y=input('请输入数据 ');  
n=length(y);  
yy=ones(n,1);  
yy(1)=y(1);  
for i=2:n  
 yy(i)=yy(i-1)+y(i);  
end  
B=ones(n-1,2);  
for i=1:(n-1)  
 B(i,1)=-(yy(i)+yy(i+1))/2;  
 B(i,2)=1;  
end  
BT=B';  
for j=1:n-1  
 YN(j)=y(j+1);  
end  
YN=YN';  
A=inv(BT\*B)\*BT\*YN;  
a=A(1);  
u=A(2);  
t=u/a;  
i=1:n+2;  
yys(i+1)=(y(1)-t).\*exp(-a.\*i)+t;  
yys(1)=y(1);  
for j=n+2:-1:2  
 ys(j)=yys(j)-yys(j-1);  
end  
x=1:n;  
xs=2:n+2;  
yn=ys(2:n+2);  
plot(x,y,'^r',xs,yn,'\*-b');  
det=0;  
  
sum1=0;  
sumpe=0;  
for i=1:n  
 sumpe=sumpe+y(i);  
end  
pe=sumpe/n;  
for i=1:n;  
 sum1=sum1+(y(i)-pe).^2;  
end  
s1=sqrt(sum1/n);  
sumce=0;  
for i=2:n  
 sumce=sumce+(y(i)-yn(i));  
end  
ce=sumce/(n-1);  
sum2=0;  
for i=2:n;  
 sum2=sum2+(y(i)-yn(i)-ce).^2;  
end  
s2=sqrt(sum2/(n-1));  
c=(s2)/(s1);  
disp(['后验差比值为：',num2str(c)]);  
if c<0.35  
 disp('系统预测精度好')  
else if c<0.5  
 disp('系统预测精度合格')  
 else if c<0.65  
 disp('系统预测精度勉强')  
 else  
 disp('系统预测精度不合格')  
 end  
 end  
end  
   
disp(['下个拟合值为 ',num2str(ys(n+1))]);  
disp(['再下个拟合值为',num2str(ys(n+2))]);

## 4.聚类算法

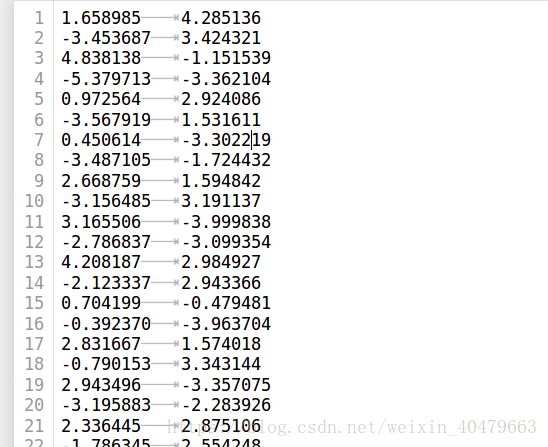
就是将数据分为k类，我们这里用kmeans，

比如将数据分成4类：，这里的样本以前都是黑色的，我们认为设定k=4，就是分为4类以后得到的分类结果



from numpy import \*  
import time  
import matplotlib.pyplot as plt  
   
   
# calculate Euclidean distance  
def euclDistance(vector1, vector2):  
 return sqrt(sum(power(vector2 - vector1, 2)))  
   
# init centroids with random samples  
def initCentroids(dataSet, k):  
 numSamples, dim = dataSet.shape  
 centroids = zeros((k, dim))  
 for i in range(k):  
 index = int(random.uniform(0, numSamples))  
 centroids[i, :] = dataSet[index, :]  
 return centroids  
   
# k-means cluster  
def kmeans(dataSet, k):  
 numSamples = dataSet.shape[0]  
 # first column stores which cluster this sample belongs to,  
 # second column stores the error between this sample and its centroid  
 clusterAssment = mat(zeros((numSamples, 2)))  
 clusterChanged = True  
   
 ## step 1: init centroids  
 centroids = initCentroids(dataSet, k)  
   
 while clusterChanged:  
 clusterChanged = False  
 ## for each sample  
 for i in xrange(numSamples):  
 minDist = 100000.0  
 minIndex = 0  
 ## for each centroid  
 ## step 2: find the centroid who is closest  
 for j in range(k):  
 distance = euclDistance(centroids[j, :], dataSet[i, :])  
 if distance < minDist:  
 minDist = distance  
 minIndex = j  
   
 ## step 3: update its cluster  
 if clusterAssment[i, 0] != minIndex:  
 clusterChanged = True  
 clusterAssment[i, :] = minIndex, minDist\*\*2  
   
 ## step 4: update centroids  
 for j in range(k):  
 pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]  
 centroids[j, :] = mean(pointsInCluster, axis = 0)  
   
 print 'Congratulations, cluster complete!'  
 return centroids, clusterAssment  
   
# show your cluster only available with 2-D data  
def showCluster(dataSet, k, centroids, clusterAssment):  
 numSamples, dim = dataSet.shape  
 if dim != 2:  
 print "Sorry! I can not draw because the dimension of your data is not 2!"  
 return 1  
   
 mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']  
 if k > len(mark):  
 print "Sorry! Your k is too large! please contact Zouxy"  
 return 1  
   
 # draw all samples  
 for i in xrange(numSamples):  
 markIndex = int(clusterAssment[i, 0])  
 plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])  
   
 mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']  
 # draw the centroids  
 for i in range(k):  
 plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize = 12)  
   
 plt.show()  
  
#测试代码：  
  
  
from numpy import \*  
import time  
import matplotlib.pyplot as plt  
   
## step 1: load data  
print "step 1: load data..."  
dataSet = []  
##################################修改##########################  
fileIn = open('E:/Python/Machine Learning in Action/testSet.txt')  
for line in fileIn.readlines():  
 lineArr = line.strip().split('\t')  
 dataSet.append([float(lineArr[0]), float(lineArr[1])])  
   
## step 2: clustering...  
print "step 2: clustering..."  
dataSet = mat(dataSet)  
k = 4  
centroids, clusterAssment = kmeans(dataSet, k)  
   
## step 3: show the result  
print "step 3: show the result..."  
showCluster(dataSet, k, centroids, clusterAssment)

数据文件长这个样子



## 5.蒙特卡洛

## 6.模拟退火

功能：

寻找全局最优值（寻找最小值或者最大值）

例题：

TSP最短路径

基本思想：

是贪心算法，在搜索过程中加入随机扰动，如果找到一个更接近最优值的点，接受该解。否则，以一定的概率接收一个比当前解要差的解，防止陷入局部最优解，**做到更加接近最优值。**模拟退火过程，初始温度最高，随着搜索的过程，温度越低，内能减小，搜索范围趋于稳定。

算法步骤：（以最小值为例）

**Step1.** 设置初始温度,当前迭代步数 ，起点

**Step2.** 得到新解 其中 在[dmin,dmax]之间

**Step3.** ，我中是优化目标

**Step4.** 若 则接受该点，否则，以概率接受该点；并降温。

**Step4.** 若 则接受该点，否则，以概率接受该点(最大值)

这样一直重复2,3,4一直达到阈值温度结束

function main  
clc  
clear;  
  
%范围  
section\_l = 0;  
section\_h = 9;  
  
%绘制函数图像  
draw\_base();  
  
%初始温度，停止温度与降温系数  
tmp = 1e5;  
tmp\_min = 1e-3;  
alpha = 0.98;  
  
%生成初始随机解  
x\_old = (section\_h - section\_l) \* rand() + section\_l;  
x\_new = x\_old;  
s\_old = val(x\_old);  
s\_new = s\_old;  
  
text\_lt = text(0, 0, 'Init');  
  
%计数器  
counter = 0;  
  
%退火的主体部分，一个循环  
while(tmp > tmp\_min) %阈值温度  
 %随机扰动  
 delta = (rand() - 0.5) \* 3;  
 x\_new = x\_old + delta;  
 %扰动的值小于一半的区间范围时，可以用这种办法防止新值超出区间范围  
 if(x\_new < section\_l || x\_new > section\_h)  
 x\_new = x\_new - 2 \* delta;  
 end  
   
 s\_new = val(x\_new);  
  
 %求差值，这里是找最大值而非最小值，所以不是s\_new - s\_old 3  
 dE = s\_old - s\_new;  
   
 %判断，是否接受该点 4   
 j = judge(dE, tmp);   
 if(j)  
 s\_old = s\_new;  
 x\_old = x\_new;  
 end  
   
 %只有当dE < 0的情况下才降温   
 if(dE < 0)  
 delete(text\_lt);  
 hold on, scatter(x\_old, s\_old);  
 hold on, text\_lt = text(0.3, 21, ['tmp: ', num2str(tmp)]);  
 pause(0.01);  
 %上面是绘图的代码，下面降温  
 tmp = tmp \* alpha;  
 else  
 counter = counter + 1;  
 end  
   
 %当接受更差的解的概率太小时，若又长时间找不到更优的解，那么退出循环，结束算法  
 if(counter > 10000)  
 break;  
 end  
   
end  
end  
function [y] = val(x)  
 y = x + 10 \* sin(5 \* x) + 7 \* cos(4 \* x);  
end  
  
function draw\_base()  
 X = 0: 0.01:9;  
 M = val(X);  
 plot(X, M);  
end  
function [y] = judge(dE, t)  
 if(dE < 0)  
 y = 1;  
 else  
 d = exp(-(dE / t));  
 if(d > rand)  
 y = 1;  
 else  
 y = 0;  
 end  
 end  
end

https://zhuanlan.zhihu.com/p/33184423

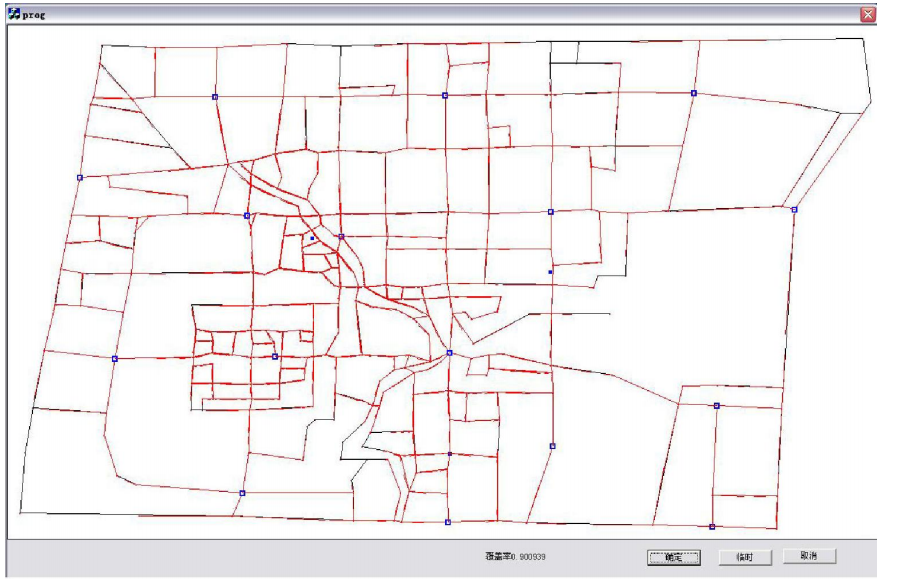
优缺点：

温度管理问题是一个难以处理的问题，实际应用中，由于必须考虑计算复杂度的切实可行性等问题，很容易受到参数的影响

具有局部搜索能力强、运行时间短的优点

例题： 110警车配置

思路：图论问题，简化问题，将图结点保留，边信息保留，由图转化成邻接矩阵的问题



## 7.遗传算法

功能：也是寻找最优值方案的算法

所以求最大值的过程就转化成一个“袋鼠跳”的过程。\*\*

下面介绍介绍“袋鼠跳”的几种方式。

* 爬山算法：一只袋鼠朝着比现在高的地方跳去。它找到了不远处的最高的山峰。但是这座山不一定是最高峰。这就是爬山算法，它不能保证局部最优值就是全局最优值。
* 模拟退火：袋鼠喝醉了。它随机地跳了很长时间。这期间，它可能走向高处，也可能踏入平地。但是，它渐渐清醒了并朝最高峰跳去。这就是模拟退火算法。
* 遗传算法：有很多袋鼠，它们降落到喜玛拉雅山脉的任意地方。这些袋鼠并不知道它们的任务是寻找珠穆朗玛峰。但每过几年，就在一些海拔高度较低的地方射杀一些袋鼠。于是，不断有袋鼠死于海拔较低的地方，而越是在海拔高的袋鼠越是能活得更久，也越有机会生儿育女。就这样经过许多年，这些袋鼠们竟然都不自觉地聚拢到了一个个的山峰上，可是在所有的袋鼠中，只有聚拢到珠穆朗玛峰的袋鼠被带回了美丽的澳洲。

https://blog.csdn.net/JX\_Cesare/article/details/81268771

https://www.jianshu.com/p/ae5157c26af9

基本思想：适者生存（接近最优值的），劣者淘汰（每隔一段时间淘汰掉一些结点）

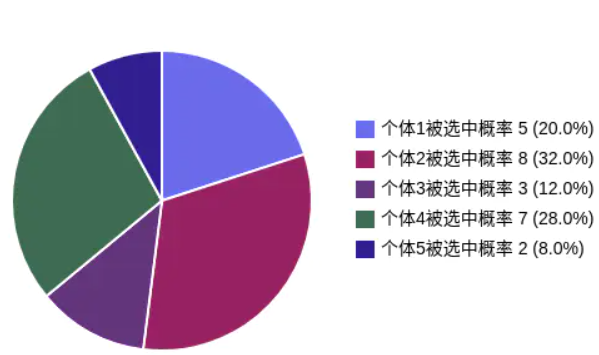
并遵循遗传中的基因遗传，变异，交配产生后代

h函数还是没有太看懂

算法步骤：

1. 编码（设计基因片段）
2. 设计适应度函数，一个目标函数
3. 选择（轮盘赌：根据自身的优势把所有种群放在0-1之间选择，随机竞争，最佳保留等算法）
4. 遗传（染色体交叉）基因片段的交换
5. 变异

轮盘赌：



function main()  
clear;  
clc;  
%种群大小  
popsize=100;  
%二进制编码长度  
chromlength=10;  
%交叉概率  
pc = 0.6;  
%变异概率  
pm = 0.001;  
%初始种群  
pop = initpop(popsize,chromlength);  
   
for i = 1:100  
 %计算适应度值（函数值）  
 objvalue = cal\_objvalue(pop);  
 fitvalue = objvalue;  
 %选择操作  
 newpop = selection(pop,fitvalue);  
 %交叉操作  
 newpop = crossover(newpop,pc);  
 %变异操作  
 newpop = mutation(newpop,pm);  
 %更新种群  
 pop = newpop;  
 %寻找最优解  
 [bestindividual,bestfit] = best(pop,fitvalue);  
 x2 = binary2decimal(bestindividual);  
 x1 = binary2decimal(newpop);  
 y1 = cal\_objvalue(newpop);  
 if mod(i,10) == 0  
 figure;  
 fplot('10\*sin(5\*x)+7\*abs(x-5)+10',[0 10]);  
 hold on;  
 plot(x1,y1,'\*');  
 title(['迭代次数为n=' num2str(i)]);  
 %plot(x1,y1,'\*');  
 end  
end  
fprintf('The best X is --->>%5.2f\n',x2);  
fprintf('The best Y is --->>%5.2f\n',bestfit);

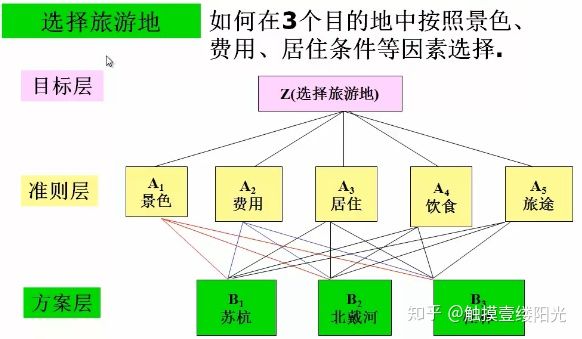
## 8.粒子群算法

## 9.种群竞争：

## 10.排队论：

## 11.层次分析：

做决策：



## 12.多元回归：

## 13.主成分分析：

对于二维数据而言：

|  |  |
| --- | --- |
|  |  |
| 原始数据 | 将原始数据求质点的数据分布 |

这样求出来的数据，便于以下的计算

x,y轴上的方差与协方差公式：

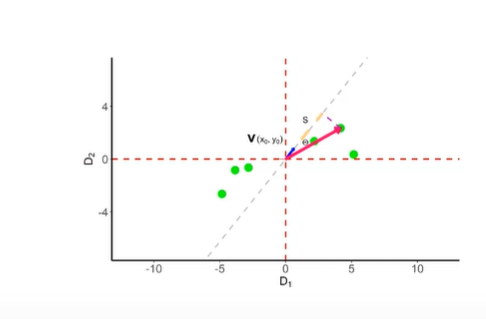
**实际上可以表示为：**

其中，是以质心为（0,0）的处理后的数据 ，形状为 4\*2

我们的目标就是重新寻找一个方向，使得这些点让在这个方向上分散得最开。

也就是说，在这个方向上，点到中心原点的方差最大。

构造 目标函数以及 约束函数，利用拉格朗日定理,求解：



求最值之后会得到一个特征值公式：

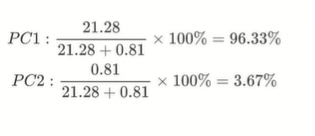
然后特征值分解会得到两个相互独立的向量（线性代数有求解过程）

|  |  |
| --- | --- |
|  |  |
| 求解的两个向量 | 表示咱图中 |

的含义：带入到公式（2）此时，Cv= 带入可得，由于是一个常数，可以提出来，

所以

最后得到的两个方向：信息集中到了一个方向里面，因此主成分分析降维技术，将信息少的维度省去



例题：



clc  
clear all  
A=xlsread('这里换成你自己的数据根目录，例如：D:\资料库区\大三上\HUAWEI\MATLAB\主成分分析.xls','B3:I17');  
%得到的数据矩阵的行数和列数  
a=size(A,1);  
b=size(A,2);  
%数据的标准化处理:得到标准化后的矩阵SA  
for i=1:b  
 SA(:,i)=(A(:,i)-mean(A(:,i)))/std(A(:,i));  
end  
%计算系数矩阵:CM  
CM=corrcoef(SA);  
%计算CM的特征值和特征向量  
[V,D]=eig(CM);  
%将特征值按降序排列到DS中  
for j=1:b  
 DS(j,1)=D(b+1-j,b+1-j);  
end  
%计算贡献率  
for i=1:b  
 DS(i,2)=DS(i,1)/sum(DS(:,1));%单个贡献率  
 DS(i,3)=sum(DS(1:i,1))/sum(DS(:,1));%累计贡献率  
end  
%假定主成分的信息保留率  
T=0.9;  
for k=1:b  
 if DS(k,3) >= T  
 com\_num=k;  
 break;  
 end  
end  
%提取主成分的特征向量  
for j=1:com\_num  
 PV(:,j)=V(:,b+1-j);  
end  
%计算主成分得分  
new\_score=SA\*PV;  
for i=1:a  
 total\_score(i,1)=sum(new\_score(i,:));  
 total\_score(i,2)=i;  
end  
%强主成分得分与总分放到同一个矩阵中  
result\_report=[new\_score,total\_score];  
%按总分降序排列  
result\_report=sortrows(result\_report,-4);  
%输出结果  
disp('特征值、贡献率、累计贡献率：')  
DS  
disp('信息保留率T对应的主成分数与特征向量：')  
com\_num  
PV  
disp('主成分得分及排序（按第4列的总分进行降序排列，前3列为个各成分得分，第5列为企业编号）')  
result\_report  
  
可以用下方数据训练一下

## 14.神经网络

**神经网络的普通应用：回归与分类**

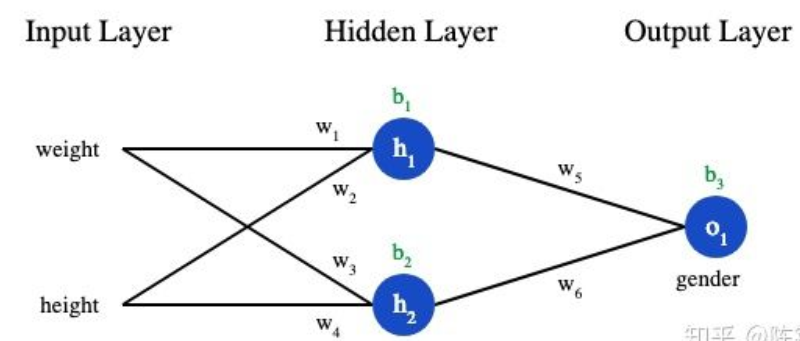
### 神经网络（2,2,1）结构

import numpy as np  
#这是一层神经网络的构造：  
def sigmoid(x):  
 # Our activation function: f(x) = 1 / (1 + e^(-x))  
 return 1 / (1 + np.exp(-x))  
  
class Neuron:  
 def \_\_init\_\_(self, weights, bias):  
 self.weights = weights  
 self.bias = bias  
  
 def feedforward(self, inputs):  
 # Weight inputs, add bias, then use the activation function  
 total = np.dot(self.weights, inputs) + self.bias  
 return sigmoid(total)  
  
weights = np.array([0, 1]) # w1 = 0, w2 = 1  
bias = 4 # b = 4  
n = Neuron(weights, bias)  
  
x = np.array([2, 3]) # x1 = 2, x2 = 3  
print(n.feedforward(x)) # 0.9990889488055994

0.9990889488055994

import numpy as np  
  
# ... code from previous section here  
  
class OurNeuralNetwork:  
 '''  
 A neural network with:  
 - 2 inputs  
 - a hidden layer with 2 neurons (h1, h2)  
 - an output layer with 1 neuron (o1)  
 Each neuron has the same weights and bias:  
 - w = [0, 1]  
 - b = 0  
 '''  
 def \_\_init\_\_(self):  
 weights = np.array([0, 1])  
 bias = 0  
  
 # The Neuron class here is from the previous section  
 self.h1 = Neuron(weights, bias)  
 self.h2 = Neuron(weights, bias)  
 self.o1 = Neuron(weights, bias) #初始化  
  
 def feedforward(self, x):  
 out\_h1 = self.h1.feedforward(x) #前向传播  
 out\_h2 = self.h2.feedforward(x)  
  
 # The inputs for o1 are the outputs from h1 and h2  
 out\_o1 = self.o1.feedforward(np.array([out\_h1, out\_h2]))  
  
 return out\_o1  
  
network = OurNeuralNetwork()  
x = np.array([2, 3])  
print(network.feedforward(x)) # 0.7216325609518421

0.7216325609518421



import numpy as np  
  
def sigmoid(x):  
 # Sigmoid activation function: f(x) = 1 / (1 + e^(-x))  
 return 1 / (1 + np.exp(-x))  
  
def deriv\_sigmoid(x):  
# Derivative of sigmoid: f'(x) = f(x) \* (1 - f(x))  
 fx = sigmoid(x)  
 return fx \* (1 - fx)  
  
def mse\_loss(y\_true, y\_pred):  
 # y\_true and y\_pred are numpy arrays of the same length.  
 return ((y\_true - y\_pred) \*\* 2).mean()  
  
class OurNeuralNetwork: #这个神经网络只是用于有2两个输入，一个隐藏层（含有2个结点），一个输出的简单的神经网络  
  
 def \_\_init\_\_(self):  
 # 权重，Weights  
 self.w1 = np.random.normal()  
 self.w2 = np.random.normal()  
 self.w3 = np.random.normal()  
 self.w4 = np.random.normal()  
 self.w5 = np.random.normal()  
 self.w6 = np.random.normal()  
  
 # 截距项，Biases  
 self.b1 = np.random.normal()  
 self.b2 = np.random.normal()  
 self.b3 = np.random.normal()  
  
 def feedforward(self, x):  
 # x is a numpy array with 2 elements.  
 h1 = sigmoid(self.w1 \* x[0] + self.w2 \* x[1] + self.b1)  
 h2 = sigmoid(self.w3 \* x[0] + self.w4 \* x[1] + self.b2)  
 o1 = sigmoid(self.w5 \* h1 + self.w6 \* h2 + self.b3)  
 return o1  
  
 def train(self, data, all\_y\_trues): #l=里面涉及到反向梯度更新传播算法  
 '''  
 - data is a (n x 2) numpy array, n = # of samples in the dataset.  
 - all\_y\_trues is a numpy array with n elements.  
 Elements in all\_y\_trues correspond to those in data.  
 '''  
 learn\_rate = 0.1  
 epochs = 1000 # number of times to loop through the entire dataset  
  
 for epoch in range(epochs):  
   
 for x, y\_true in zip(data, all\_y\_trues):  
   
   
 # --- Do a feedforward (we'll need these values later) 每一次的前向传播，可以用feedforward代替  
 sum\_h1 = self.w1 \* x[0] + self.w2 \* x[1] + self.b1  
 h1 = sigmoid(sum\_h1)  
  
 sum\_h2 = self.w3 \* x[0] + self.w4 \* x[1] + self.b2  
 h2 = sigmoid(sum\_h2)  
  
 sum\_o1 = self.w5 \* h1 + self.w6 \* h2 + self.b3  
 o1 = sigmoid(sum\_o1)  
 y\_pred = o1  
 '''  
 h1,h2,o1 =OurNeuralNetwork.feedforward(data) #类的使用还不太会  
 y\_pred = o1  
 '''  
  
 # --- Calculate partial derivatives.  
 # --- Naming: d\_L\_d\_w1 represents "partial L / partial w1"  
 d\_L\_d\_ypred = -2 \* (y\_true - y\_pred)  
  
 # Neuron o1  
 d\_ypred\_d\_w5 = h1 \* deriv\_sigmoid(sum\_o1)  
 d\_ypred\_d\_w6 = h2 \* deriv\_sigmoid(sum\_o1)  
 d\_ypred\_d\_b3 = deriv\_sigmoid(sum\_o1)  
  
 d\_ypred\_d\_h1 = self.w5 \* deriv\_sigmoid(sum\_o1)  
 d\_ypred\_d\_h2 = self.w6 \* deriv\_sigmoid(sum\_o1)  
  
 # Neuron h1  
 d\_h1\_d\_w1 = x[0] \* deriv\_sigmoid(sum\_h1)  
 d\_h1\_d\_w2 = x[1] \* deriv\_sigmoid(sum\_h1)  
 d\_h1\_d\_b1 = deriv\_sigmoid(sum\_h1)  
  
 # Neuron h2  
 d\_h2\_d\_w3 = x[0] \* deriv\_sigmoid(sum\_h2)  
 d\_h2\_d\_w4 = x[1] \* deriv\_sigmoid(sum\_h2)  
 d\_h2\_d\_b2 = deriv\_sigmoid(sum\_h2)  
  
 # --- Update weights and biases  
 # Neuron h1  
 self.w1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w1  
 self.w2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_w2  
 self.b1 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h1 \* d\_h1\_d\_b1  
  
 # Neuron h2  
 self.w3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w3  
 self.w4 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_w4  
 self.b2 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_h2 \* d\_h2\_d\_b2  
  
 # Neuron o1  
 self.w5 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w5  
 self.w6 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_w6  
 self.b3 -= learn\_rate \* d\_L\_d\_ypred \* d\_ypred\_d\_b3  
  
 # --- Calculate total loss at the end of each epoch  
 if epoch % 10 == 0:  
 y\_preds = np.apply\_along\_axis(self.feedforward, 1, data)  
 loss = mse\_loss(all\_y\_trues, y\_preds)  
 print(1)  
 print("Epoch %d loss: %.3f" % (epoch, loss))  
  
# Define dataset  
data = np.array([  
 [-2, -1], # Alice  
 [25, 6], # Bob  
 [17, 4], # Charlie  
 [-15, -6], # Diana  
])  
all\_y\_trues = np.array([  
 1, # Alice  
 0, # Bob  
 0, # Charlie  
 1, # Diana  
])  
  
# Train our neural network!  
network = OurNeuralNetwork()  
print(network.train(data, all\_y\_trues))

