《人工智能及应用》

代码实验课

授课人：张 鑫　zhangxin@uestc.edu.cn
专　业：机器人工程

提纲

1、搜索的概念

2、盲目搜索

3、启发式搜索

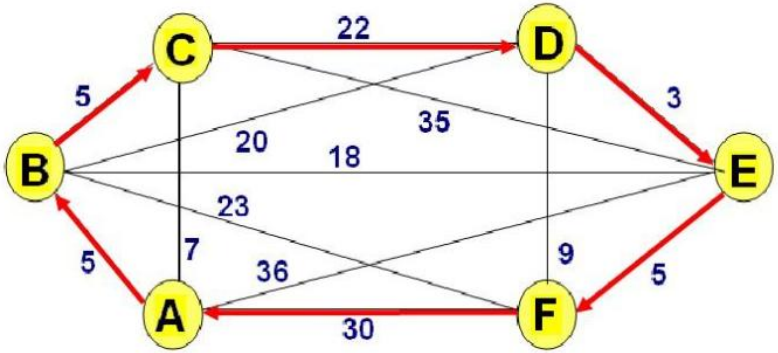
4、智能搜索

5、博弈搜索

1、蚁群算法

1.1概念与起源

旅行商问题（简称TSP），也称货郎担问题或旅行推销员问题，是运筹学中一个著名的问题，其一般提法为：有一个旅行商从城市1出发，需要到城市2、3、…、n去推销货物，最后返回城市1，若任意两个城市间的距离已知，则该旅行商应如何选择其最佳行走路线？



TSP问题可以表示为一个N个城市的有向图G=（N，A）。

其中N={1, 2,..., n}　　A={(i, j) | i, j∈ *N*}

城市之间距离 $(d_{ij})_{nxn}$

目标函数 $f(w)=\sum_{l=1}^{n} d_{i_l i_{l+1}}$

其中$w=(i_1, i_2, ..., i_n)$为城市1, 2, ..., n的一个排列，$i_{n+1}=i_1$

1、蚁群算法

1. 2蚁群算法的模型

m 是蚁群中蚂蚁的数量

$d_{xy}(x, y = 1, \ldots, n)$ 表示节点(城市) 和节点(城市) 之间的距离

$\eta_{xy}(t)$ 表示能见度函数，等于距离的倒数，即 $\eta_{xy}(t) = \frac{1}{d_{xy}}$

$b_x(t)$ 表示t时刻位于节点x的蚂蚁的个数， $m = \sum_{x=1}^{n} b_x(t)$

$\tau_{xy}(t)$ 表示t时刻在xy连线上残留的信息素，初始时刻，各条路径上的信息素相等即 $\tau_{xy}(0) = C(const)$

如果C太小，算法容易早熟，蚂蚁会很快全部集中到一条局部最优路径上。如果C太大，信息素对搜索方向的指导作用太低，也会影响算法性能。

1、蚁群算法

1. 2蚁群算法的模型

$P_{xy}^{k}(t)$ 表示在t时刻蚂蚁 k 选择从元素(城市) x 转移到元素(城市) y 的概率，也称为随机比例规则。

信息素 $\tau_{xy}(t)$

局部启发信息 $\eta_{xy}(t)$

共同决定 → $P_{xy}^{k}(t)$

$\eta_{xy}(t)$ 表示能见度函数，等于距离的倒数，即 $\eta_{xy}(t)=\frac{1}{d_{xy}}$

1、蚁群算法

1. 2蚁群算法的模型

随机比例规则 $P_{xy}^{k}(t)$ 表示如下：

$$P_{xy}^{k}(t)=\begin{cases}\dfrac{\left|\tau_{xy}(t)\right|^{\alpha}\left|\eta_{xy}(t)\right|^{\beta}}{\sum_{y\in allowed_k(x)}\left|\tau_{xy}(t)\right|^{\alpha}\left|\eta_{xy}(t)\right|^{\beta}} & if \quad y\in allowed_k(x)\\ 0 & 其他\end{cases}$$

α=0：随机贪婪算法

β=0：正反馈启发式算法

其中：

$allowed_k(x)=\{0,1,...,n-1\}-tabu_k(x)$ 表示蚂蚁k下一步允许选择的节点(城市)

$tabu_k(x)$ $(k=1,2,...,m)$ 记录蚂蚁k当前所走过的城市

α 和β为两常数，分别是信息素和能见度的加权值。

1、蚁群算法

1.2蚁群算法的模型

用参数ρ（$0<\rho<1$）表示信息素消逝程度，蚂蚁完成一次循环，各路径上信息素浓度消散规则为：

$$\tau_{xy}(t+1)=(1-\rho)\tau_{xy}(t)+\Delta\tau_{xy}(t)$$

蚁群的信息素浓度更新规则为：

$$\Delta\tau_{xy}(t)=\sum_{k=1}^{m}\Delta\tau_{xy}^{k}(t)$$

$$\Delta\tau_{xy}^{k}(t)=\begin{cases}Q/L_{xy}^{k} & \text{蚂蚁k经过xy路径(有多种定义)}\\ 0\end{cases}$$

其中：$\tau_{xy}(t)$为当前路径上的信息素

$\Delta\tau_{xy}(t)$为路径（x，y）上信息素的增量

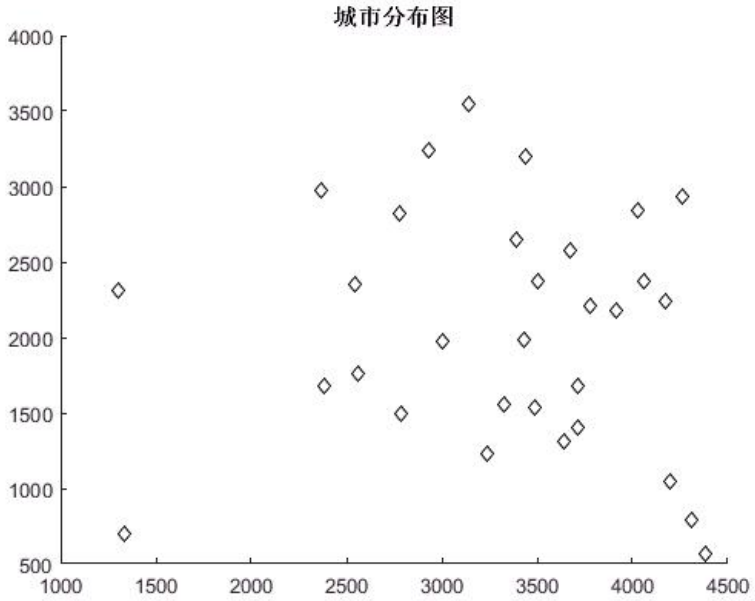
$\Delta\tau_{xy}^{k}(t)$第k只蚂蚁留在路径（x，y）上的信息素的增量

Q 为信息素常数，L_{xy}^{k} 为第k只蚂蚁本次迭代走过的总路径长度

1、蚁群算法

1.2蚁群算法的模型

在一张地图上有n个城市，一名推销员需要不重复的一次走过n个城市进行推销，求解他按照怎样的路径，从才能使走过的距离最短。



1、蚁群算法

1. 2蚁群算法的模型

例子：四个城市的TSP问题，A, B, C, D 。现在要从城市A出发，最后又回到A，期间B, C, D 都必须并且只能经过一次，使代价最小。

$$D=(d_{ij})=\begin{pmatrix}0&3&1&2\\3&0&5&4\\1&5&0&2\\2&4&2&0\end{pmatrix}$$

1. 初始化：
假设共m=3只蚂蚁，参数α=1，β=2，ρ=0.5, τ0=0.3，Q=1

2. 为每个蚂蚁随机选择出发城市，假设蚂蚁1选择A，蚂蚁2选择B，蚂蚁3选择D

3. 为每个蚂蚁选择下一个访问城市，以蚂蚁1为例：当前城市i=A，可访问城市集合allowed(i)={B, C, D}

1、蚁群算法

1.2蚁群算法的模型

```
% 背景：在一张地图上有n个城市，一名推销员需要不重复的一次走过n个城市进行推销，
% 求解他按照怎样的路径，从才能使走过的距离最短。
clear
close all;
clc;
 % 城市坐标
C = [1304 2312;
3639 1315;
4177 2244;
3712 1399;
3488 1535;
3326 1556;
3238 1229;
4196 1044;
4312 790;
4386 570;
3007 1970;
2562 1756;
2788 1491;
2381 1676;
1332 695;
3715 1678;
3918 2179;
4061 2370;
3780 2212;
3676 2578;
4029 2838;
4263 2931;
3429 1980;
3507 2376;
3394 2643;
3439 3201;
2935 3240;
3140 3550;
2545 2357;
2778 2826;
2370 2975];
figure(1);
```

```
[M,N] = size(C);
% M为问题的规模 M个城市
distance = zeros(M,M); % 用来记录任意两个城市之间的距离
% 求任意两个城市之间的距离
for m=1:M
for n=1:M
distance(m,n) = sqrt(sum((C(m,:)-C(n,:)).^2));
end
end
m = 50; % 蚂蚁的个数 一般取10-50
alpha = 1; % 信息素的重要程度 一般取【1,4】
beta = 5; % 启发式英子的重要程度 一般取【3,5】
rho = 0.25; % 信息素蒸发系数
G = 150;
Q = 100; % 信息素增加系数
Eta = 1./distance; % 启发式因子
Tau = ones(M,M); % 信息素矩阵 存储着每两个城市之间的信息素的数值
Tabu = zeros(m,M); % 禁忌表，记录每只蚂蚁走过的路程
gen = 1;
R_best = zeros(G,M); % 各代的最佳路线
L_best = inf.*ones(G,1); % 每一代的最佳路径的长度 初始假设为无穷大
```

1、蚁群算法

1. 2蚁群算法的模型

3. 为每个蚂蚁选择下一个访问城市，以蚂蚁1为例：
当前城市i=A，可访问城市集合allowed(i)={B, C, D}

计算蚂蚁1访问各个城市的概率：

$$A \Rightarrow \begin{cases} B: \tau_{AB}^{a} \times \eta_{AB}^{\beta} = 0.3^{1} \times (1/3)^{2} = 0.033 \\ C: \tau_{AC}^{a} \times \eta_{AC}^{\beta} = 0.3^{1} \times (1/1)^{2} = 0.300 \\ D: \tau_{AD}^{a} \times \eta_{AD}^{\beta} = 0.3^{1} \times (1/2)^{2} = 0.075 \end{cases}$$

$$p(B) = 0.033/(0.033+0.3+0.075) = 0.081$$

$$p(C) = 0.3/(0.033+0.3+0.075) = 0.74$$

$$p(D) = 0.075/(0.033+0.3+0.075) = 0.18$$

$$P_{xy}^{k}(t) = \begin{cases} \dfrac{\left|\tau_{xy}(t)\right|^{\alpha} \left|\eta_{xy}(t)\right|^{\beta}}{\sum\limits_{y \in allowed_{k}(x)} \left|\tau_{xy}(t)\right|^{\alpha} \left|\eta_{xy}(t)\right|^{\beta}} \\ 0 \end{cases}$$

使用轮盘赌法选择下一个城市

1、蚁群算法

1.2蚁群算法的模型

```
% 开始迭代计算
while gen<G
% 将m只蚂蚁放到n个城市上
random_pos = [];
for i=1:(ceil(m/M)) % m只蚂蚁随即放到M座城市
random_pos = [random_pos,randperm(M)]; % random_pos=[1~31 + 1~31] 将每只蚂蚁放到随机的城市 在random_pos 中
end
Tabu(:,1) = (random_pos(1,1:m))'; % 第一次迭代每只蚂蚁的禁忌表

for i=2:M % 从第二个城市开始
for j=1:m % 每只蚂蚁
visited = Tabu(j,1:(i-1)); % 在访问第i个城市的时候，第j个蚂蚁访问过的城市
% visited=visited(1,:);
unvisited = zeros(1,(M+1-i)); % 待访问的城市
visit_P = unvisited; % 蚂蚁j访问剩下的城市的概率
count = 1;
for k=1:M % 这个循环是找出未访问的城市
if isempty(find(visited==k,1)) %还没有访问过的城市 如果成立。则证明第k个城市没有访问过
unvisited(count) = k;
count = count+1;
end
end
% 计算待选择城市的概率
for k=1:length(unvisited) % Tau(visited(end),unvisited(k))访问过的城市的最后一个与所有未访问的城市之间的信息素
visit_P(k) = ((Tau(visited(end),unvisited(k)))^alpha)*(Eta(visited(end),unvisited(k))^beta);
end
visit_P = visit_P/sum(visit_P); % 访问每条路径的概率的大小
```

1、蚁群算法

1. 2蚁群算法的模型

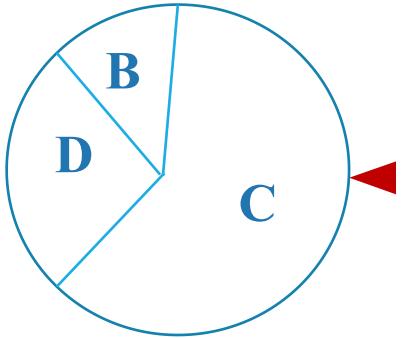
3. 为每个蚂蚁选择下一个访问城市，以蚂蚁1为例：

当前城市i=A，可访问城市集合allowed(i)={B, C, D}

$$p(B) = 0.033/(0.033+0.3+0.075) = 0.081$$

$$p(C) = 0.3/(0.033+0.3+0.075) = 0.74$$

$$p(D) = 0.075/(0.033+0.3+0.075) = 0.18$$



轮盘赌法：在选择路径时概率大的路径被选中的概率大，同时概率小的路径也有可能被选择，而不是直接选择概率大的路径，避免所有蚂蚁在这里都做出同样的选择，导致算法失去随机性。

B：0 < rand < 0.081

D：0.081 < rand < 0.081+0.18

C：0.081+0.18 < rand < 1

假设产生的随机数rand=0.05，蚂蚁1选择城市B

同样假设蚂蚁2选择城市D，蚂蚁3选择城市A。

1、蚁群算法

1. 2蚁群算法的模型

4. 对于蚂蚁1，

当前城市i=B，已访问集合tabu(i)={A,B}，可访问集合allowed(i)={ C, D}

计算蚂蚁1访问C，D城市的概率：

$$B \Rightarrow \begin{cases} C: \tau_{BC}^{a} \times \eta_{BC}^{\beta} = 0.3^{1} \times (1/5)^{2} = 0.012 \\ D: \tau_{BD}^{a} \times \eta_{BD}^{\beta} = 0.3^{1} \times (1/4)^{2} = 0.019 \end{cases}$$

$$p(C) = 0.012/(0.012+0.019) = 0.39$$

$$p(D) = 0.019/(0.012+0.019) = 0.61$$

$$P_{xy}^{k}(t) = \begin{cases} \dfrac{\left|\tau_{xy}(t)\right|^{\alpha} \left|\eta_{xy}(t)\right|^{\beta}}{\sum\limits_{y \in allowed_{k}(x)} \left|\tau_{xy}(t)\right|^{\alpha} \left|\eta_{xy}(t)\right|^{\beta}} \\ 0 \end{cases}$$

假设产生的随机数rand=0.1，蚂蚁1选择城市C。

同样假设蚂蚁2选择C，蚂蚁3选择D。

1、蚁群算法

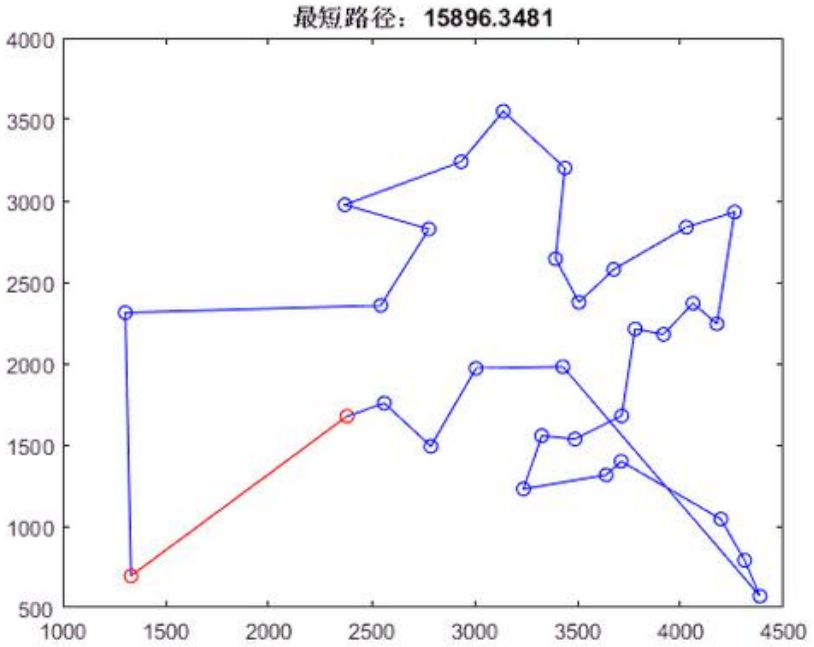
1.2蚁群算法的模型

```
% 按照概率选择下一个要访问的城市
% 这里运用轮盘赌选择方法 这里也可以选择选择概率最大的路径去走， 这里采用轮盘赌选择法。
Pcum = cumsum(visit_P);
selected = find(Pcum>=rand);
to_visited = unvisited(selected(1));
Tabu(j,i) = to_visited; % 添加到禁忌表
end
end
if gen>=2
Tabu(1,:) = R_best(gen-1,:);
end
% 记录m只蚂蚁迭代的最佳路线
L = zeros(1,m);
for i=1:m
R = Tabu(i,:);
L(i) = distance(R(M),R(1)); % 因为要走一周回到原来的地点
for j=1:(M-1)
L(i) = L(i)+distance(R(j),R(j+1));
end
end
L_best(gen) = min(L); % 记录每一代中路径的最短值
pos = find(L==L_best(gen));
R_best(gen,:) = Tabu(pos(1),:); % 最优的路径

% 更新信息素的值
Delta_Tau = zeros(M,M);
for i=1:m % m只蚂蚁
for j=1:(M-1) % M座城市
Delta_Tau(Tabu(i,j),Tabu(i,j+1)) = Delta_Tau(Tabu(i,j),Tabu(i,j+1)) + Q/L(i); % m只蚂蚁的信息素累加 j
end
Delta_Tau(Tabu(i,M),Tabu(i,1)) = Delta_Tau(Tabu(i,M),Tabu(i,1)) + Q/L(i);
end
Tau = (1-rho).*Tau+Delta_Tau; % 更新路径上的信息素含量
% 禁忌表清零
Tabu = zeros(m,M);
```

1、蚁群算法

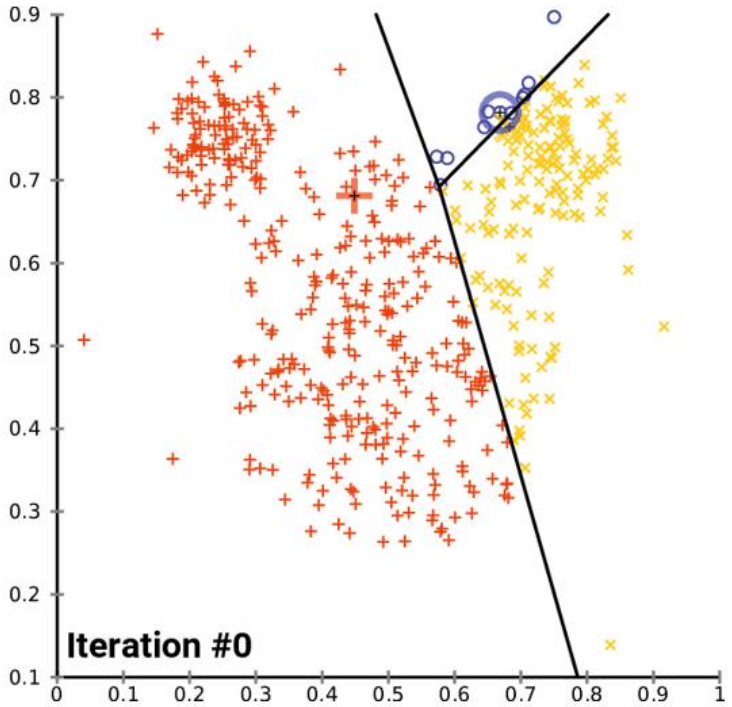
1.2蚁群算法的模型

```
for i=1:(M-1)
plot([C(R_best(gen,i),1),C(R_best(gen,i+1),1)],[C(R_best(gen,i),2),C(R_best(gen,i+1),2)],'bo-');
hold on;
end
plot([C(R_best(gen,n),1),C(R_best(gen,1),1)],[C(R_best(gen,n),2),C(R_best(gen,1),2)],'ro-');
title(['最短路径: ',num2str(L_best(gen))]);
hold off;
pause(0.05);
gen = gen+1;
end
figure(2);
plot(L_best);
title('路径长度变化曲线');
xlabel('迭代次数');
ylabel('路径长度数值');
```



2、K均值聚类

2. 2算法



输入：聚类个数K

Step1: 设有K个聚类$C=\{C_1, C_2, ..., C_K\}$, 随机选择K个样本分别作为K个聚类的中心点$\{m_1, m_2, ..., m_K\}$

Step2:对每个样本x_j，将其分配到距离最近的聚类中心所对应的聚类

Step3:对第i个聚类，重新计算其中心

$$m'_i = \frac{1}{n_i} \sum_{x_j \in C_i} x_j$$

其中n_i为聚类C_i中的样本个数

Step4: 重复step2和step3，直到满足收敛条件（达到指定的迭代次数，聚类中心不再变化等）

输出：聚类结果

2、K均值聚类

2.2算法

- 相似度定义：用**欧式距离**描述数据之间的相似性。
- 有n个数据$\{x_1, x_2, \cdots, x_n\}$，$(1 \leqslant i \leqslant n)$
- 每个数据是m维的，$x_i = \begin{bmatrix} x_{i1} & x_{i2} & \cdots & x_{im} \end{bmatrix}^T$
- 两个m维数据之间的**欧氏距离**定义为：

$$d(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{im} - x_{jm})^2}$$

2、K均值聚类

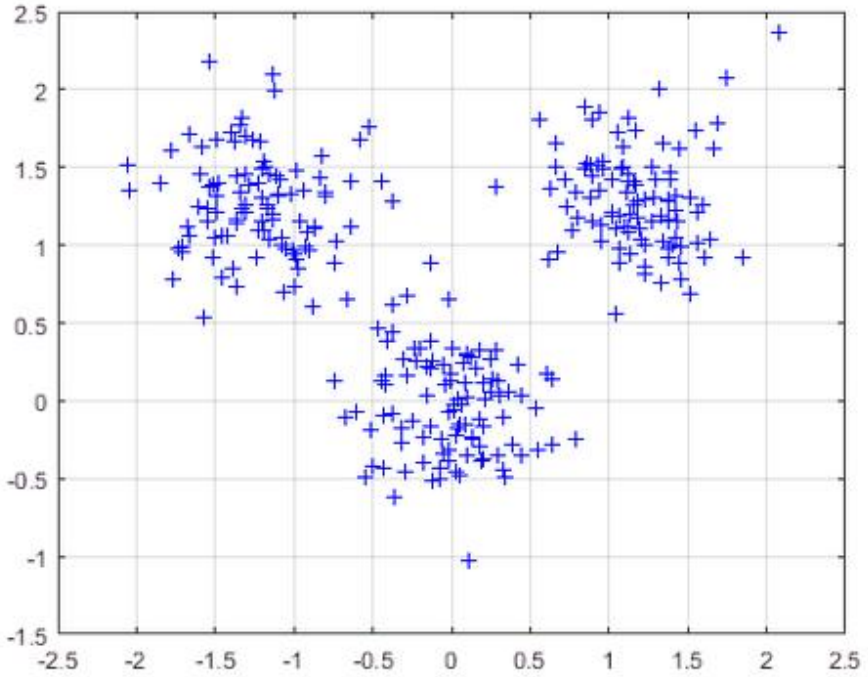
2.2算法

- 需要事先确定**聚类数目K**，很多时候我们并不知道数据应被聚类的数目
- 需要**初始化聚类质心**，初始化聚类中心对聚类结果有较大的影响
- 算法是迭代执行，时间开销非常大
- 欧氏距离假设数据每个维度之间的重要性是一样的

2、K均值聚类

2.2算法

```
clear
close all;
clc;
% 第一组数据
mu1=[0 0 ];  %均值(是需要生成的数据的均值)
S1=[.1 0 ;0 .1];  %协方差(需要生成的数据的自相关矩阵（相关系数矩阵）)
data1=mvnrnd(mu1,S1,100);    %产生高斯分布数据
%第二组数据
mu2=[1.25 1.25 ];
S2=[.1 0 ;0 .1];
data2=mvnrnd(mu2,S2,100);
% 第三组数据
mu3=[-1.25 1.25 ];
S3=[.1 0 ;0 .1];
data3=mvnrnd(mu3,S3,100);
% 显示数据
plot(data1(:,1),data1(:,2),'b+');
hold on;%不覆盖原图，要关闭则使用hold off;
plot(data2(:,1),data2(:,2),'b+');
plot(data3(:,1),data3(:,2),'b+');
grid on;%显示表格
```



2、K均值聚类

2.2算法

```
%  三类数据合成一个不带标号的数据类
data=[data1;data2;data3];
N=3;%设置聚类数目
[m,n]=size(data);%表示矩阵data大小，m行n列
pattern=zeros(m,n+1);%生成0矩阵
center=zeros(N,n);%初始化聚类中心
pattern(:,1:n)=data(:,:);
for x=1:N
    center(x,:)=data( randi(300,1),:);%第一次随机产生聚类中心
end
while 1 %循环迭代每次的聚类簇;
    distence=zeros(1,N);%最小距离矩阵
    num=zeros(1,N);%聚类簇数矩阵
    new_center=zeros(N,n);%聚类中心矩阵

    for x=1:m
        for y=1:N
            distence(y)=norm(data(x,:)-center(y,:));%计算到每个类的距离
        end
        [~, temp]=min(distence);%求最小的距离
        pattern(x,n+1)=temp;%划分所有对象点到最近的聚类中心；标记为1,2,3;
    end
```

2、K均值聚类

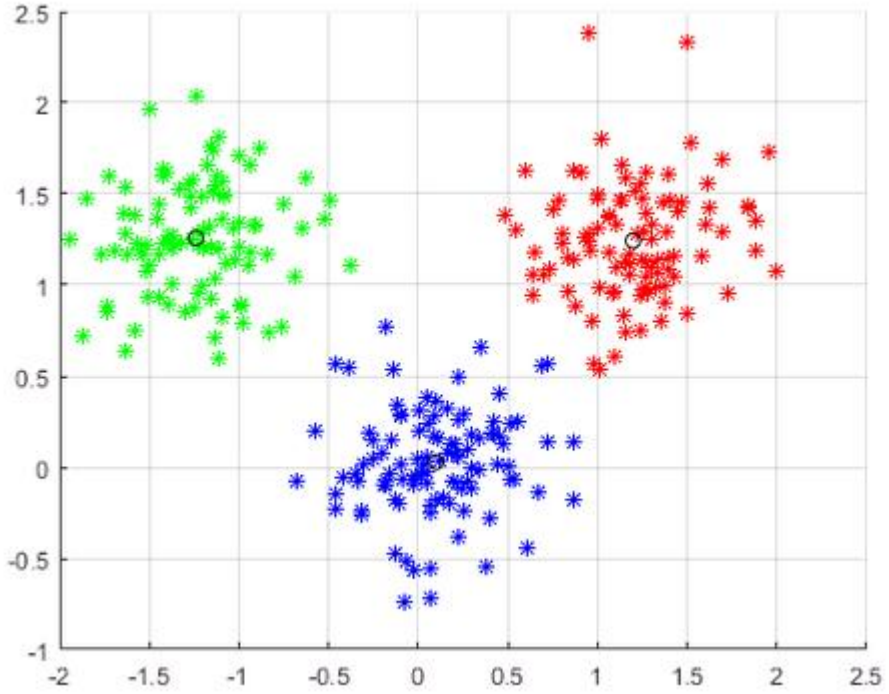
2.2算法

```
    k=0;
    for y=1:N
        for x=1:m
            if pattern(x,n+1)==y
                new_center(y,:)=new_center(y,:)+pattern(x,1:n);
                num(y)=num(y)+1;
            end
        end
        new_center(y,:)=new_center(y,:)/num(y);%求均值，即新的聚类中心；
        if norm(new_center(y,:)-center(y,:))<0.1%检查集群中心是否已收敛。如果是则终止。
            k=k+1;
        end
    end
    if k==N
        break;
    else
        center=new_center;
    end
end
[m, n]=size(pattern);
```

2、K均值聚类

2. 2算法

```
%最后显示聚类后的数据
figure;
hold on;
for i=1:m
    if pattern(i,n)==1
        plot(pattern(i,1),pattern(i,2),'r*');
        plot(center(1,1),center(1,2),'ko');%用小圆圈标记中心点;
    elseif pattern(i,n)==2
        plot(pattern(i,1),pattern(i,2),'g*');
        plot(center(2,1),center(2,2),'ko');
    elseif pattern(i,n)==3
        plot(pattern(i,1),pattern(i,2),'b*');
        plot(center(3,1),center(3,2),'ko');
    elseif pattern(i,n)==4
        plot(pattern(i,1),pattern(i,2),'y*');
        plot(center(4,1),center(4,2),'ko');
    else
        plot(pattern(i,1),pattern(i,2),'m*');
        plot(center(4,1),center(4,2),'ko');
    end
end
grid on;
```

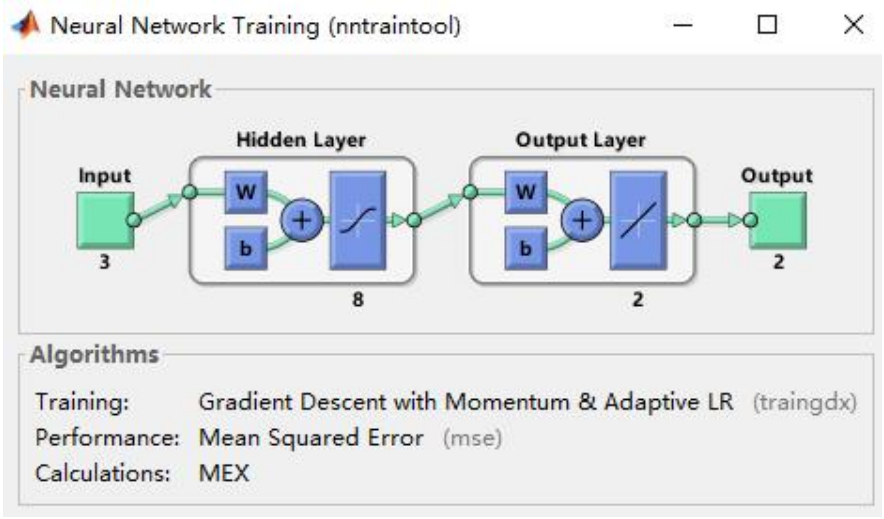


3、BP神经网络

3. 2算法

（1）构建一个3层BP神经网络对该地区公路运力进行预测：输入层结点数为3个，隐含层结点数为8，隐含层的激活函数为‘tansig’；输出层结点数为2个，输出层的激活函数为‘purelin’。

（2）采用梯度下降动量和自适应lr算法&traingdx%训练BP网络，目标误goal=1×10-3，学习率lr=0.035，最大迭代次数epochs=2000。



3、BP神经网络

3. 2算法

```
% 拟合的历年公路客运量曲线和历年公路货运量曲线分别如图所示。
%%%%%%%%%%%%%%%%%%%%%%运用BP网络预测数据%%%%%%%%%%%%%%%%%%%%%%
clear                          %清除所有变量
close  all;                        %清图
clc;                               %清屏
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%原始数据%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%人数(单位: 万人)%%%%%%%%%%%%%%%%%%%%%%%%%
sqrs=[20.55 22.44 25.37 27.13 29.45 30.10 30.96 34.06 36.42 38.09...
    39.13 39.99 41.93 44.59 47.30 52.89 55.73 56.76 59.17 60.63];
%%%%%%%%%%%%%%%%%%%%%%%机动车数(单位: 万辆)%%%%%%%%%%%%%%%%%%%%%%%
sqjdcs=[0.6 0.75 0.85 0.9 1.05 1.35 1.45 1.6 1.7 1.85 2.15 2.2...
    2.25 2.35 2.5 2.6 2.7 2.85 2.95 3.1];
%%%%%%%%%%%%%%%%%%%%%%%公路面积(单位: 万平方公里)%%%%%%%%%%%%%%%%%%%%%%
sqglmj=[0.09 0.11 0.11 0.14 0.20 0.23 0.23 0.32 0.32 0.34 0.36...
    0.36 0.38 0.49 0.56 0.59 0.59 0.67 0.69 0.79];
%%%%%%%%%%%%%%%%%%%%%%%%公路客运量(单位: 万人)%%%%%%%%%%%%%%%%%%%%%%%%
glkyl=[5126 6217 7730 9145 10460 11387 12353 15750 18304 19836 ...
    21024 19490 20433 22598 25107 33442 36836 40548 42927 43462];
%%%%%%%%%%%%%%%%%%%%%%%%公路货运量(单位: 万吨)%%%%%%%%%%%%%%%%%%%%%%%%
glhyl=[1237 1379 1385 1399 1663 1714 1834 4322 8132 8936 11099 ...
    11203 10524 11115 13320 16762 18673 20724 20803 21804];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%输入数据矩阵%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
p=[sqrs;sqjdcs;sqglmj];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%目标数据矩阵%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
t=[glkyl;glhyl];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%原始样本归一化%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[P,PSp] = mapminmax(p);
[T,PSt] = mapminmax(t);
%%%%%%%%%%%%%%%%%%%%%%创建一个新的神经网络%%%%%%%%%%%%%%%%%%%%%%%%
net=newff(P,T,8,{'tansig','purelin'},'traingdx');
```

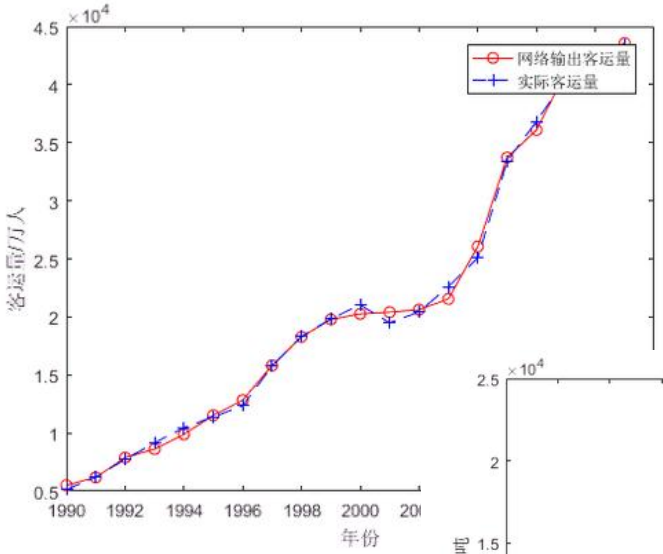
```
traingd: 梯度下降算法
traingdm: 带动量的梯度下降算法
traingda: 学习率变化的梯度下降算法
traingdx: 学习率变化带动量的梯度下降算法
trainrp: RPROP算法，内存需求小，适用于大型网络
trainoss: OneStep Secant Algorithm，计算量与内存需求较小，适用于大型网络
```

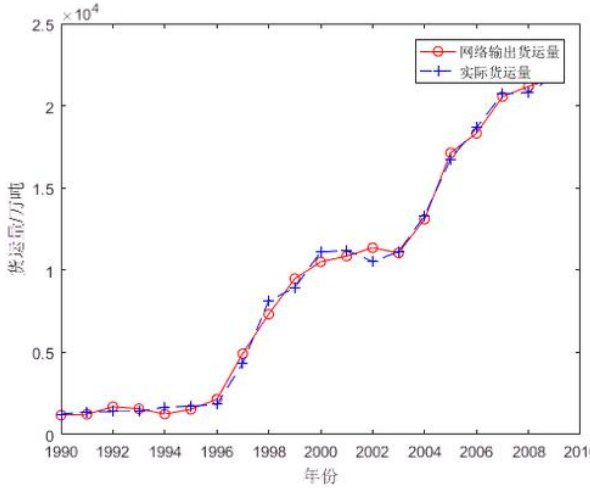
```
logsig: 对数S型函数
tansig: 正切S型函数
purelin: 线性型函数
```

3、BP神经网络

3. 2算法

```
%%%%%%%%%%%%%%%%%%%%%%%%设置训练参数%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
net.trainParam.show = 50;                %显示中间结果的周期
net.trainParam.lr = 0.035;               %学习率
net.trainParam.epochs = 1000;            %最大迭代次数
net.trainParam.goal = 1e-3;              %目标误差
net.divideFcn = '';  %清除样本数据分为训练集、验证集和测试集命令
%%%%%%%%%%%%%%%%%%%%调用 TRAINGDM 算法训练 BP 网络%%%%%%%%%%%%%%%%%%%%
[net,tr]=train(net,P,T);
%%%%%%%%%%%%%%%%%%%%%%对 BP 网络进行仿真%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = sim(net,P);
a=mapminmax('reverse',A,PSt);
%%%%%%%%%%%%%%%%%%%%%%优化后输入层权值和阈值%%%%%%%%%%%%%%%%%%%%%%%%
inputWeights=net.IW{1,1};
inputbias=net.b{1};
%%%%%%%%%%%%%%%%%%%%%%优化后网络层权值和阈值%%%%%%%%%%%%%%%%%%%%%%%%
layerWeights=net.LW{2,1};
layerbias=net.b{2};
%%%%%%%%%%%%%%%%%%%%%%%%%%时间轴刻度%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x=1990:2009;
%%%%%%%%%%%%%%%%%%%%%%%%网络输出客运量%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
newk=a(1,:);
%%%%%%%%%%%%%%%%%%%%%%%%网络输出货运量%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
newh=a(2,:);
%%%%%%%%%%%%%%%%%%%%%%绘值公路客运量对比图%%%%%%%%%%%%%%%%%%%%%%%%%
figure
plot(x,newk,'r-o',x,glkyl,'b--+')
legend('网络输出客运量','实际客运量');
xlabel('年份');ylabel('客运量/万人');
%%%%%%%%%%%%%%%%%%%%%%%绘制公路货运量对比图%%%%%%%%%%%%%%%%%%%%%%%%%
figure
plot(x,newh,'r-o',x,glhyl,'b--+')
legend('网络输出货运量','实际货运量');
xlabel('年份');ylabel('货运量/万吨');
%%%%%%%%%%%%%%%%%%%%%%%利用训练好的网络进行预测%%%%%%%%%%%%%%%%%%%%%%
```





3、BP神经网络

3.2算法

```
%%%%%%%%%%%%%%%%%%%%%%%%利用训练好的网络进行预测%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%2010年和2011年的相关数据%%%%%%%%%%%%%%%%%%%%%%%%
pnew=[73.39 75.55;3.9 4.1;0.98 1.02];  %给出2010年及2011年人数，机动车数，公路面积
SamNum=size(pnew,2);
%%%%%%%%%%%%%利用原始输入数据的归一化参数对新数据进行归一化%%%%%%%%%%%%%
pnewn=mapminmax('apply',pnew,PSp);
%%%%%%%%%%%%%%%%%%%%%%%%%%隐含层输出预测结果%%%%%%%%%%%%%%%%%%%%%%%%%%%
HiddenOut=tansig(inputWeights*pnewn+repmat(inputbias,1,SamNum));
%%%%%%%%%%%%%%%%%%%%%%%%%%输出层输出预测结果%%%%%%%%%%%%%%%%%%%%%%%%%%%
anewn=purelin(layerWeights*HiddenOut+repmat(layerbias,1,SamNum));
%%%%%%%%%%%%%%%%把网络预测得到的数据还原为原始的数量级%%%%%%%%%%%%%%%%
anew=mapminmax('reverse',anewn,PSt);
fprintf('2010年公路客运量为%g亿人,2010年公路货运量为%g亿吨\n',anew(1,1)/10000,anew(2,1)/10000)
fprintf('2011年公路客运量为%g亿人,2011年公路货运量为%g亿吨\n',anew(1,2)/10000,anew(2,2)/10000)
```

```
2010年公路客运量为4.39485亿人,2010年公路货运量为2.15288亿吨
2011年公路客运量为4.39332亿人,2011年公路货运量为2.1478亿吨
```

4、Q-Learning

4. 2算法

```
%% 基于Q-learning算法的机器人路径规划系统
clear
close all;
clc;
%% 首先创造一个机器人运动的环境
% n是该运动的运动环境的矩阵environment(n,n)的
n = 20;
% 新建一个全为1的n*n维environment矩阵
environment = ones(n,n);

%下面设置环境中的障碍物，将其在矩阵中标为值-100
environment(2,2:5)=-100;
environment(5,3:5)=-100;
environment(4,11:15)=-100;
environment(2,13:17)=-100;
environment(7,14:18)=-100;
environment(3:10,19)=-100;
environment(15:18,19)=-100;
environment(3:10,19)=-100;
environment(3:10,7)=-100;
environment(9:19,2)=-100;
environment(15:17,7)=-100;
environment(10,3:7)=-100;
environment(13,5:8)=-100;
environment(6:8,4)=-100;
environment(13:18,4)=-100;
environment(6:16,10)=-100;
environment(19:20,10)=-100;
environment(17,13:17)=-100;
environment(18,6:11)=-100;
environment(10:17,13)=-100;
environment(10,13:17)=-100;
environment(14,15:19)=-100;
nvironment(7,12)=-100;
```

```
 1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
 1 -100 -100 -100 -100 -100   20    1    1    1    1    1 -100 -100 -100 -100 -100    1    1    1
 1    1    1    1    1    1    1 -100    1    1    1    1    1    1    1    1    1    1    1 -100    1
 1    1    1    1    1    1    1 -100    1    1    1    1 -100 -100 -100 -100 -100    1    1    1 -100    1
 1    1 -100 -100 -100    1    1 -100    1    1    1    1    1    1    1    1    1    1    1    1 -100    1
 1    1    1 -100    1    1    1 -100    1    1 -100    1    1    1    1    1    1    1    1    1 -100    1
 1    1    1 -100    1    1    1 -100    1    1 -100    1    1    1 -100 -100 -100 -100 -100 -100 -100    1
 1 -100    1    1    1    1    1 -100    1    1 -100    1    1    1    1    1    1    1    1    1 -100    1
 1 -100 -100 -100 -100 -100 -100 -100    1    1 -100    1    1 -100 -100 -100 -100 -100    1 -100    1
 1 -100    1    1    1    1    1    1    1    1 -100    1    1 -100    1    1    1    1    1    1    1    1
 1 -100    1    1    1    1    1    1    1    1 -100    1    1 -100    1    1    1    1    1    1    1    1
 1 -100    1 -100 -100 -100 -100 -100    1    1 -100    1    1 -100    1    1    1    1    1    1    1    1
 1 -100    1 -100    1    1    1    1    1    1 -100    1    1 -100    1    1 -100 -100 -100 -100 -100    1
 1 -100    1 -100    1    1    1 -100    1    1 -100    1    1 -100    1    1    1    1    1    1 -100    1
 1 -100    1 -100    1    1    1 -100    1    1 -100    1    1 -100    1    1    1    1    1    1 -100    1
 1 -100    1 -100    1    1    1 -100    1    1    1    1    1 -100 -100 -100 -100 -100 -100    1 -100    1
 1 -100    1 -100    1 -100 -100 -100 -100 -100 -100 -100    1    1    1    1    1    1    1    1   20    1
 1 -100    1    1    1    1    1    1    1    1 -100    1    1    1    1    1    1    1    1    1    1    1
 1    1    1    1    1    1    1    1    1    1 -100    1    1    1    1    1    1    1    1    1    1    1
```


```
% 指定机器人运动的起点和终点的x,y值，对应的起点和终点在矩阵中分别标为1和10
% 自动初始化确定起点和终点的x,y坐标
Start_point_x = 2;
Start_point_y = 6;
End_point_x = 18;
End_point_y = 19;
```

4、Q-Learning

4. 2算法

```
%% 构造奖励矩阵reward
% 有八种可能的运动状态，如下：
% * 向上运动      :  (i-n)；% * 向下运动      :  (i+n)
% * 向左运动      :  (i-1)；% * 向右运动      :  (i+1)
% * 向右下运动  :  (i+n+1)；% * 向左下运动  :  (i+n-1)
% * 向右上运动  :  (i-n+1)；% * 向左上运动  :  (i-n-1)
% reward矩阵初始化为0
reward=zeros(Size);
for i=1:Size
    reward(i,:) = reshape(environment',1,Size);   % 矩阵environment的元素返回到一个Size×Size的矩阵rewar
end
%将斜向运动的奖励矩阵值赋为0.7071
for i=1:Size
    for j=1:Size
        if(i+n+1<400 && reward(i,i+n+1)~=-100)
            reward(i,i+n+1) = 1/sqrt(2);
        end
        if(i+n-1>0 && i+n-1<400 && reward(i,i+n-1)~=-100)
            reward(i,i+n-1) = 1/sqrt(2);
        end
        if(i-n+1>0 && i-n+1<400 &&reward(i,i-n+1)~=-100)
            reward(i,i-n+1) = 1/sqrt(2);
        end
        if(i-n-1>0 && reward(i,i-n-1)~=-100)
            reward(i,i-n-1) = 1/sqrt(2);
        end
    end
end
for i=1:Size
    for j=1:Size
        if j~=i-n  && j~=i+n  && j~=i-1 && j~=i+1 && j~=i+n+1 && j~=i+n-1 && j~=i-n+1 && j~=i-n-1
            reward(i,j) = -Inf;
        end
    end
end
```

4、Q-Learning

4. 2算法

```
%% 通过循环迭代得到Q-learning算法的Q值表
% 设置Q-learning算法参数的gamma，alpha，循环迭代次数number和终点Goal
q = 0.5*ones(size(reward));  % 产生标准正态分布的n*n随机数矩阵
gamma = 0.9;
alpha = 0.6;
number = 80;
Goal = (End_point_x - 1)*n + End_point_y;
Min_number_of_total_steps = 400;
len = zeros(1,number); %存储每次迭代路径长度

% 循环迭代
% initial: 当前状态
% next: 下个状态
for i=1:number
    % 开始状态
    initial = 1;
    % 重复运行，直至到达goal状态
    while(1)
        % 选择状态的所有可能行动 n_actions，n_actions是一个行向量
        next_actions = find(reward(initial,:)>0);  % find()函数的基本功能是返回向量或者矩阵中不为0的元素的
        % 随机选择一个动作，并把它作为下一状态
        next = next_actions(randi([1 length(next_actions)],1,1));   % randi()函数生成从1-length(n_ac
        % 找到所有可能的动作
        next_actions = find(reward(next,:)>=0);
        % 找到最大的Q值，也就是说，为下一个行动最好的状态
        max_q = 0;
        for j=1:length(next_actions)
            max_q = max(max_q,q(next,next_actions(j)));
        end
        % 利用Bellman's equation更新Q值表
        q(initial,next) = (1-alpha)*q(initial,next) + alpha*(reward(initial,next) + gamma*max_q);
        % 检查是否到达终点
        if(initial == Goal)
            break;
        end
    % 把下一状态设置为当前状态
    initial = next;
    end
```

初始化q_π函数

循环

　初始化s为初始状态

　循环

　　采样$a \sim \epsilon - greedy_\pi(s)$

　　执行动作a，观察奖励R和下一个状态s'

　　更新$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha\left[R + \gamma \max_{a'} q_\pi(s',a') - q_\pi(s,a)\right]$

　　$s \leftarrow s'$

　直到s是终止状态

直到q_π收敛

采样策略与更新策略不同

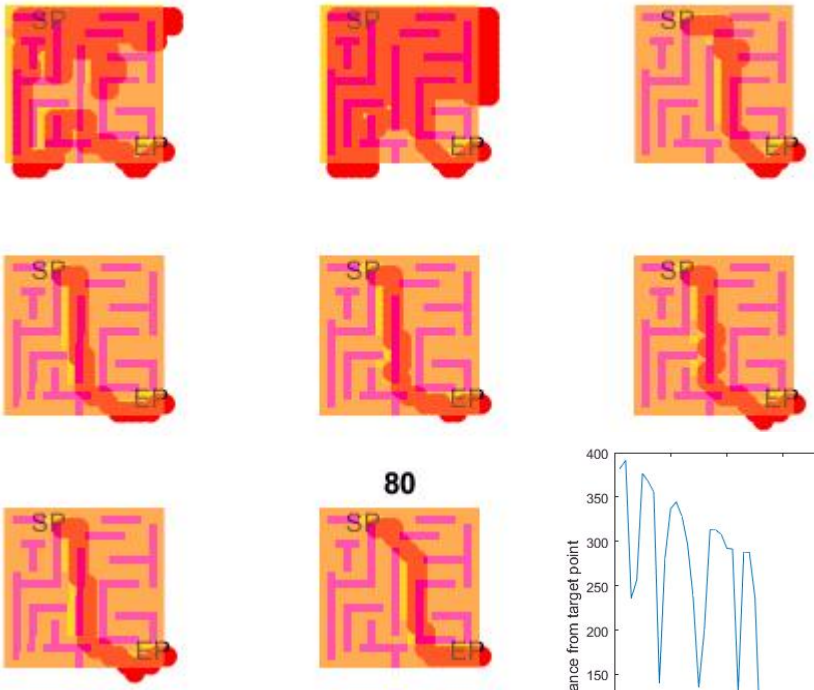
4、Q-Learning

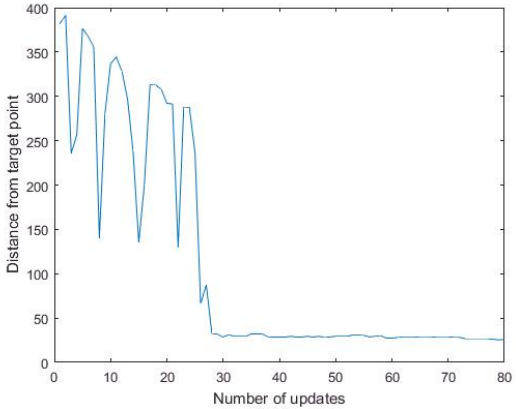
4. 2算法

```
if i<=80
    %% 下面使用Q-learning算法来进行路径规划
    start = (Start_point_x-1)*n + Start_point_y;
    path = start;
    move = 0;
    % 循环迭代直至找到终点Goal
    while(move~=Goal)
        % 从起点开始搜索
        [~,move] = max(q(start,:));
        % sort()排序函数，按降序排序
        % 消除陷入小循环的情况
        step = 2;
        while ismember(move,path)
            [~,x] = sort(q(start,:),'descend');
            move = x(step);
            step = step + 1;
        end
        % 加上下一个动作到路径中
        path = [path,move];
        start = move;
    end

    %% 输出当前机器人的最短路径和最短路径步数
    if length(path) < Min_number_of_total_steps
        Min_number_of_total_steps = length(path);
    end
    %fprintf('Final path is: %s\n',num2str(path))  % num2str()函数把数值转换成字符串，转换后使用fprintf函数进行输出
    for k=2:length(path)
        if abs(path(k)-path(k-1))==1 || abs(path(k)-path(k-1))==20
            len(i) = len(i) +1;
        else
            len(i) = len(i) + 1.4;
        end
    end
    fprintf('%d Number of total steps is : %.1f\n',i,len(i))
    % 将路径加入到新的路径矩阵pmat中
```

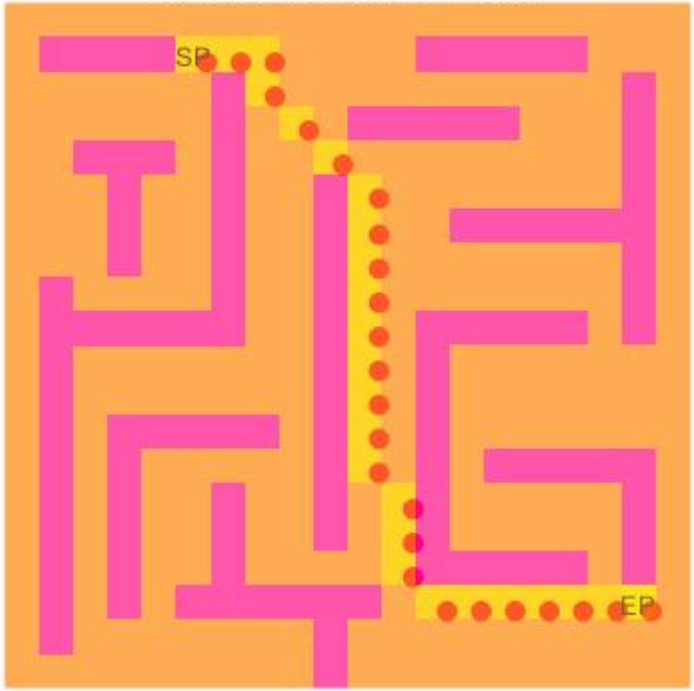
4、Q-Learning

4. 2算法





机器人最终的最短路径可视化



5、机器学习

5.1 逻辑回归算法

1. 数据集: Algerian Forest Fires Dataset(244个样本，10个属性，二分类问题)
2. 使用决策树将数据随机划分为70%的训练集、10%的验证集和20%的测试集
3. 对模型在训练集、测试集上的预测准确率进行计算并比较，分析过拟合程度

5、机器学习

5.1 逻辑回归算法

Bejaia Region Dataset

day	month	year	Temperatu	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
1	6	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
5	6	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire
6	6	2012	31	67	14	0	82.6	5.8	22.2	3.1	7	2.5	fire
7	6	2012	33	54	13	0	88.2	9.9	30.5	6.4	10.9	7.2	fire
8	6	2012	30	73	15	0	86.6	12.1	38.3	5.6	13.5	7.1	fire
9	6	2012	25	88	13	0.2	52.9	7.9	38.8	0.4	10.5	0.3	not fire
10	6	2012	28	79	12	0	73.2	9.5	46.3	1.3	12.6	0.9	not fire
11	6	2012	31	65	14	0	84.5	12.5	54.3	4	15.8	5.6	fire
12	6	2012	26	81	19	0	84	13.8	61.4	4.8	17.7	7.1	fire
13	6	2012	27	84	21	1.2	50	6.7	17	0.5	6.7	0.2	not fire
14	6	2012	30	78	20	0.5	59	4.6	7.8	1	4.4	0.4	not fire
15	6	2012	28	80	17	3.1	49.4	3	7.4	0.4	3	0.1	not fire
16	6	2012	29	89	13	0.7	36.1	1.7	7.6	0	2.2	0	not fire
17	6	2012	30	89	16	0.6	37.3	1.1	7.8	0	1.6	0	not fire
18	6	2012	31	78	14	0.3	56.9	1.9	8	0.7	2.4	0.2	not fire
19	6	2012	31	55	16	0.1	79.9	4.5	16	2.5	5.3	1.4	not fire
20	6	2012	30	80	16	0.4	59.8	3.4	27.1	0.9	5.1	0.4	not fire
21	6	2012	30	78	14	0	81	6.3	31.6	2.6	8.4	2.2	fire
22	6	2012	31	67	17	0.1	79.1	7	39.5	2.4	9.7	2.3	not fire
23	6	2012	32	62	18	0.1	81.4	8.2	47.7	3.3	11.5	3.8	fire
24	6	2012	32	66	17	0	85.9	11.2	55.8	5.6	14.9	7.5	fire
25	6	2012	31	64	15	0	86.7	14.2	63.8	5.7	18.3	8.4	fire
26	6	2012	31	64	18	0	86.8	17.8	71.8	6.7	21.6	10.6	fire
27	6	2012	34	53	18	0	89	21.6	80.3	9.2	25.8	15	fire
28	6	2012	32	55	14	0	89.1	25.5	88.5	7.6	29.7	13.9	fire
29	6	2012	32	47	13	0.3	79.9	18.4	84.4	2.2	23.8	3.9	not fire
30	6	2012	33	50	14	0	88.7	22.9	92.8	7.2	28.3	12.9	fire
1	7	2012	29	68	19	1	59.9	2.5	8.6	1.1	2.9	0.4	not fire
2	7	2012	27	75	19	1.2	55.7	2.4	8.3	0.8	2.8	0.3	not fire

5、机器学习

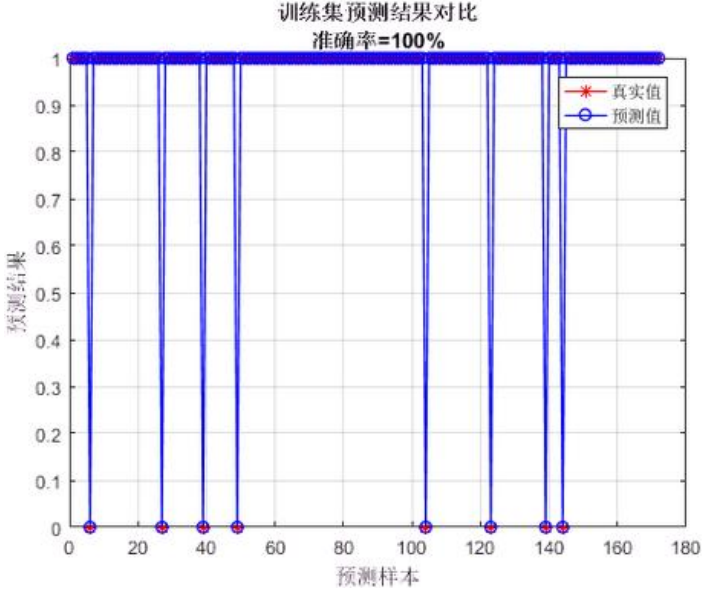
5.1 逻辑回归算法

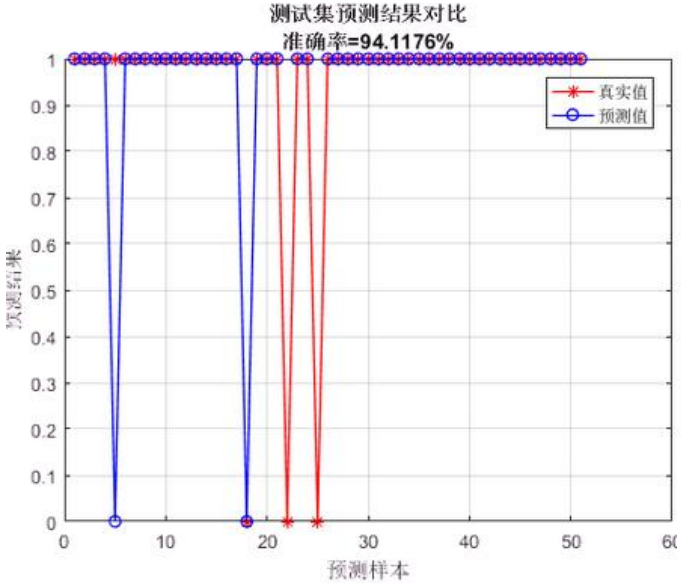
```
% 导入数据 匹配数据文件
filename = 'datafile.csv'; % Excel文件名
sheet = 1; % Excel中的工作表索引或名称
res = xlsread(filename, sheet);
% filename = 'Algerian_forest_fires_dataset_UPDATE.csv';
% res = readmatrix(filename); % 使用readmatrix读取CSV文件

% 计算数据集各部分大小
% 将数据集随机划分为70%的训练集、10%的验证集和20%的测试集
total_samples = size(res, 1);
train_size = floor(0.7 * total_samples);
val_size = floor(0.1 * total_samples);
test_size = total_samples - train_size - val_size;

% 随机打乱数据顺序
indices = randperm(total_samples);

% 划分训练集、验证集和测试集
train_indices = indices(1:train_size);
val_indices = indices(train_size+1:train_size+val_size);
test_indices = indices(train_size+val_size+1:end);

P_train = res(train_indices, 1:12)'; % 训练集的特征数据
T_train = res(train_indices, 13)';   % 训练集的标签数据
P_val = res(val_indices, 1:12)';     % 验证集的特征数据
T_val = res(val_indices, 13)';       % 验证集的标签数据
P_test = res(test_indices, 1:12)';   % 测试集的特征数据
T_test = res(test_indices, 13)';     % 测试集的标签数据

% 计算每个数据集的样本数
M = size(P_train, 2); % 训练集样本数
V = size(P_val, 2);   % 验证集样本数
N = size(P_test, 2);  % 测试集样本数
```

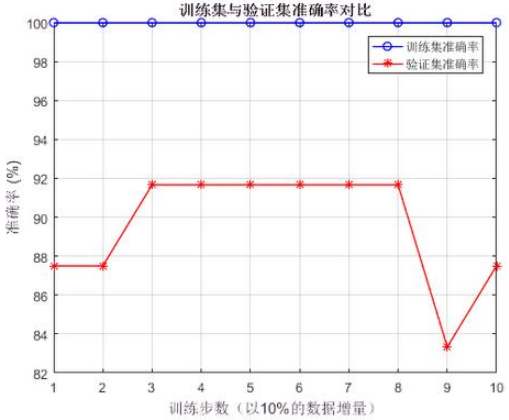
```
% 数据归一化
[p_train, ps_input] = mapminmax(P_train, 0, 1);
[p_val, ~] = mapminmax('apply', P_val, ps_input);
p_test = mapminmax('apply', P_test, ps_input);
t_train = T_train;
t_val = T_val;
t_test = T_test;

% 转置以适应模型
p_train = p_train'; p_val = p_val'; p_test = p_test';
t_train = t_train'; t_val = t_val'; t_test = t_test';

% 将标签数据二值化，确保在0到1之间
t_train_binary = t_train > 0; % 将标签大于0的数据设为1，其余为0
t_test_binary = t_test > 0; % 同上

% 训练模型
net = fitglm(p_train, t_train_binary, 'Distribution', 'binomial', 'Link', 'logit');

% 仿真测试
t_sim1 = predict(net, p_train);
t_sim2 = predict(net, p_test);

% 格式转换
T_sim1 = round(t_sim1);
T_sim2 = round(t_sim2);

% 性能评价
error1 = sum((T_sim1 == t_train_binary)) / M * 100;
error2 = sum((T_sim2 == t_test_binary)) / N * 100;
```

5、机器学习

5.1 逻辑回归算法

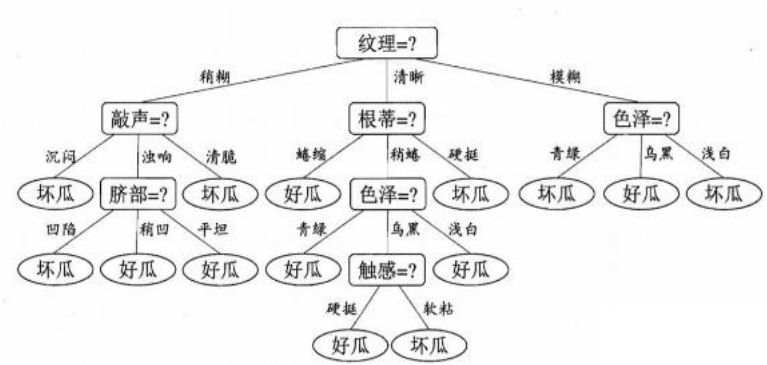






5、机器学习

5.2 随机森林算法（Random Forest，RF）

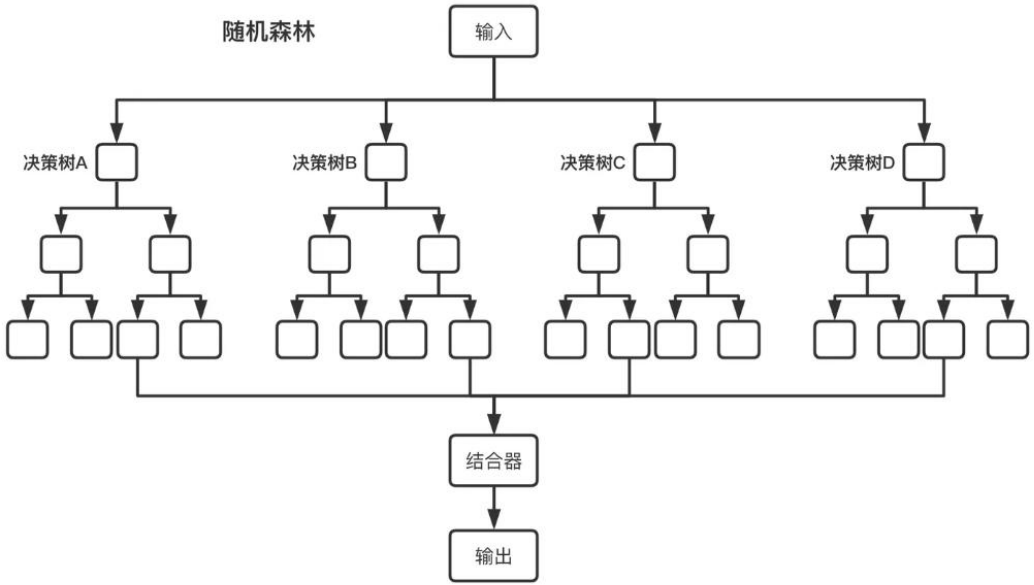


遍历随机森林的大小 K 次：

(1) 从训练集 T 中有放回抽样的方式，取样 N 次形成一个新子训练集 D;

(2) 随机选择 m 个特征，其中 m < M;

(3) 使用新的训练集 D 和 m 个特征，学习出一个完整的决策树。



5、机器学习

5.2 随机森林算法（Random Forest，RF）

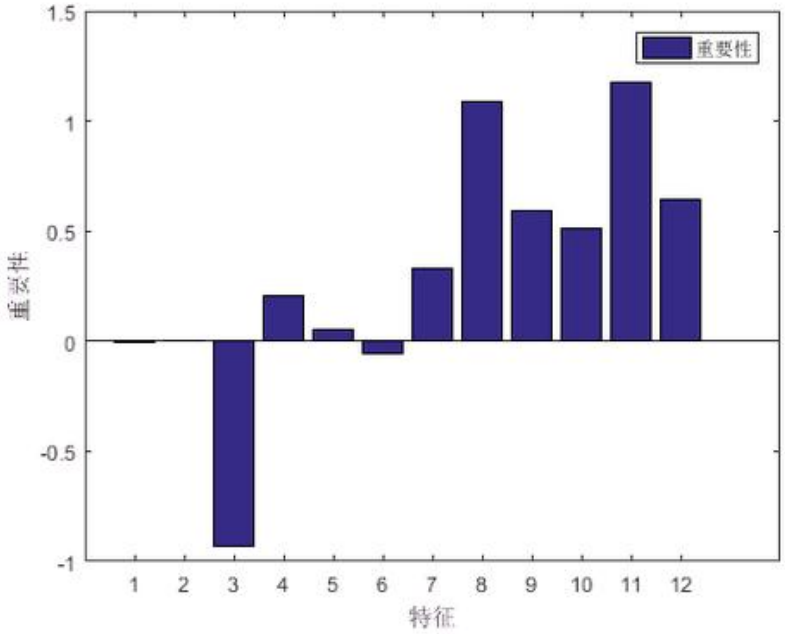
```
% 清空环境变量
warning off;             % 关闭报警信息
close all;               % 关闭开启的图窗
clear;                   % 清空变量
clc;                     % 清空命令行

% 导入数据
filename = 'datafile.csv'; % 修改文件名以匹配您的数据文件
res = readmatrix(filename); % 使用readmatrix读取CSV文件

% 计算数据集各部分大小
%将数据集随机划分为70%的训练集、10%的验证集和20%的测试集
total_samples = size(res, 1);
train_size = floor(0.7 * total_samples);
val_size = floor(0.1 * total_samples);
test_size = total_samples - train_size - val_size;

% 随机打乱数据顺序
indices = randperm(total_samples);

% 划分训练集、验证集和测试集
train_indices = indices(1:train_size);
val_indices = indices(train_size+1:train_size+val_size);
test_indices = indices(train_size+val_size+1:end);

P_train = res(train_indices, 1:12)';
T_train = res(train_indices, 13)';
P_val = res(val_indices, 1:12)';
T_val = res(val_indices, 13)';
P_test = res(test_indices, 1:12)';
T_test = res(test_indices, 13)';
```

```
% 计算每个数据集的样本数
M = size(P_train, 2); % 训练集样本数
V = size(P_val, 2);   % 验证集样本数
N = size(P_test, 2);  % 测试集样本数

% 数据归一化
[p_train, ps_input] = mapminmax(P_train, 0, 1);
[p_val, ~] = mapminmax('apply', P_val, ps_input);
p_test = mapminmax('apply', P_test, ps_input);
t_train = T_train;
t_val = T_val;
t_test = T_test;

% 转置以适应模型
p_train = p_train'; p_val = p_val'; p_test = p_test';
t_train = t_train'; t_val = t_val'; t_test = t_test';

% 训练模型
%%%%%%%%%%%% 调整参数会影响训练结果准确率%%%%%%%%%%%%%%%%%%
trees = 800;                                    % 增加决策树数量
leaf = 3;                                       % 增加最小叶子节点样本数以减少模型复杂度
MaxFeatures = floor(sqrt(size(p_train, 2)));    % 每个决策树使用的特征数量

net = TreeBagger(trees, p_train, t_train, ...
    'OOBPredictorImportance', 'on', 'Method', 'classification', ...
    'OOBPrediction', 'on', 'MinLeafSize', leaf, 'NumPredictorsToSample', MaxFeatures);
OOBPrediction = 'on';                           % 打开误差图
OOBPredictorImportance = 'on';                  % 计算特征重要性
Method = 'classification';                      % 分类还是回归
net = TreeBagger(trees, p_train, t_train, 'OOBPredictorImportance', OOBPredictorImportance, ...
      'Method', Method, 'OOBPrediction', OOBPrediction, 'minleaf', leaf);
importance = net.OOBPermutedPredictorDeltaError;  % 重要性
```

5、机器学习

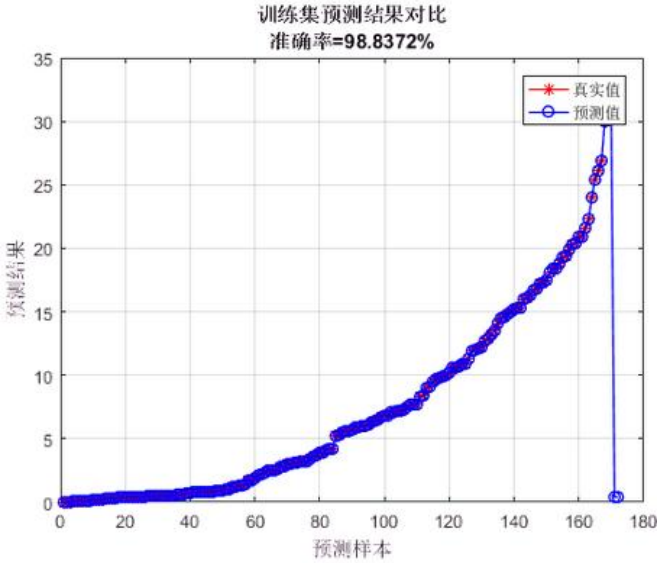
5.2 随机森林算法（Random Forest，RF）

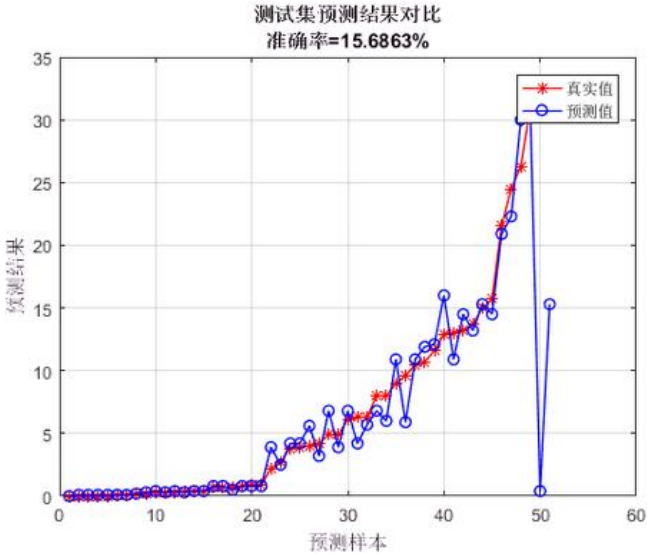
```
% 仿真测试
t_sim1 = predict(net, p_train);
t_sim2 = predict(net, p_test);

% 格式转换
T_sim1 = str2double(t_sim1);
T_sim2 = str2double(t_sim2);

% 性能评价
error1 = sum((T_sim1' == T_train)) / M * 100; % 使用正确定义的M
error2 = sum((T_sim2' == T_test)) / N * 100;  % 使用正确定义的N

% 绘制误差曲线
figure;
plot(1:trees, oobError(net), 'b-', 'LineWidth', 1);
legend('误差曲线');
xlabel('决策树数目');
ylabel('误差');
xlim([1, trees]);
grid on;

% 绘制特征重要性
figure;
bar(importance);
legend('重要性');
xlabel('特征');
ylabel('重要性');

% 数据排序
[T_train, index_1] = sort(T_train);
[T_test, index_2] = sort(T_test);

T_sim1 = T_sim1(index_1);
T_sim2 = T_sim2(index_2);
```



5、机器学习

5.2 随机森林算法（Random Forest，RF）





谢　谢