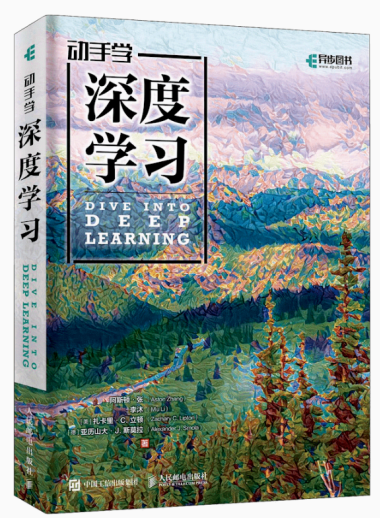
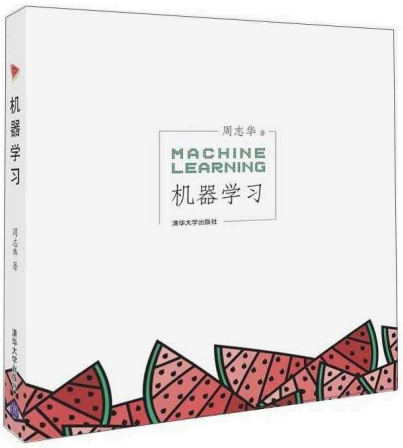
人工智能从未入门到入门

1. 阅读须知——by ZhouJH

本文档针对完全不了解人工智能的读者，旨在使之能对人工智能有个初始印象。文档中不涉及任何数学推理，只保留简单的逻辑推理，可放心食用。因为相关原理需要有一定的数学门槛，而本文仅从一个细微点切入并且点到为止，不会详尽的叙述繁多的模型和方法，这也必然导致某些结论是比较暴力的、不严谨的，如需更系统化的学习请上[b站大学]学习相应课程。



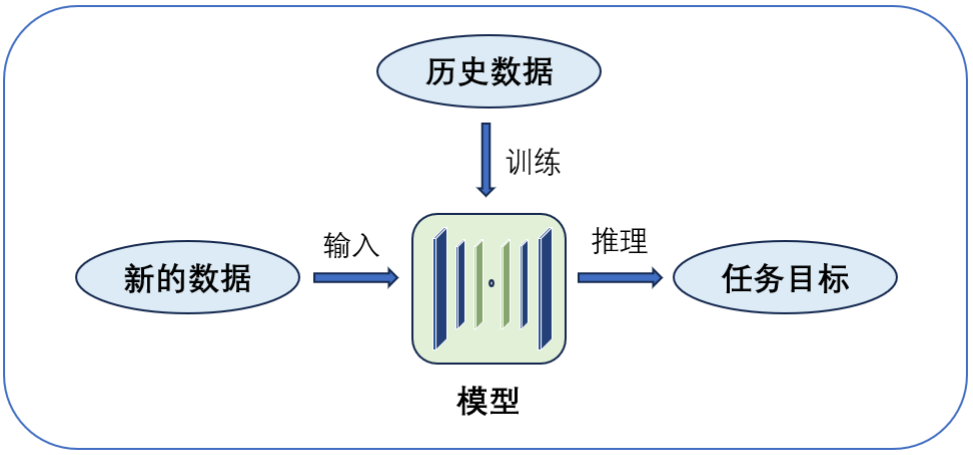


2. 一个简单的模型

2.1 一眼能看出来的$\hat{Y} = kX$

王婆最近开了个水果摊准备卖西瓜，西瓜的供货商被一个供应商垄断了。她分别去张三、李四两个人的摊位询问后得知“今早张三进货了**100斤**西瓜共花费**200元**，李四进货了**200斤**西瓜共花费**400元**”，如果王婆明天准备进货**150斤**西瓜，那么她需要准备多少的本钱来进货呢？一眼就能看出来本钱是**300元**。假设用$\hat{Y}$表示预测本钱，X表示西瓜的斤数，那么k就是本钱和斤数的规律，不言而喻$k = 2$.

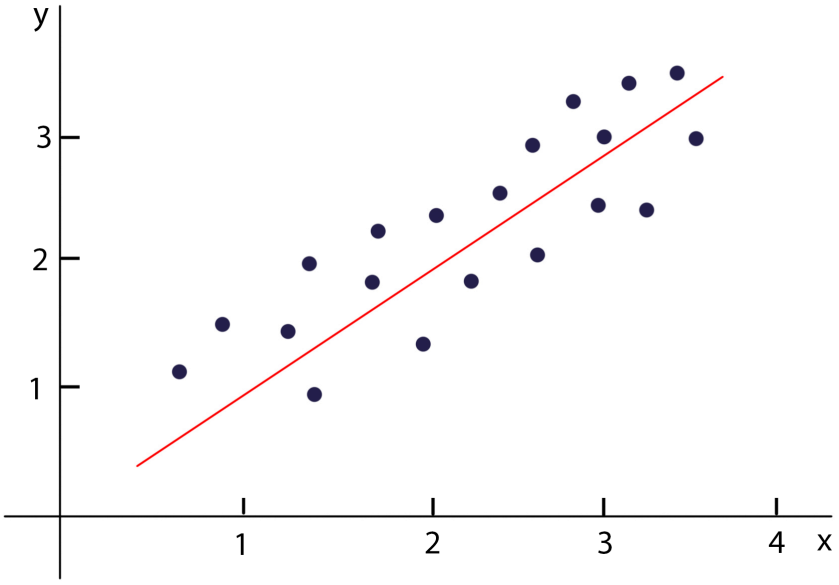
在上诉案例中，我们为什么可以仅凭**今天的数据**和直觉就可以**预测**出**明天本钱**需要多少呢，因为我们一眼就发现了西瓜的本钱是斤数的两倍这一**规律**。从已有的**经验**（已知**数据**：100斤200元、200斤400元）总结出**规律**（**模型**：$\hat{Y} = 2X$），从而**预测**（由150斤**推理**出300元）未来的花费，这个过程就是人工智能的内核。例如智驾系统中，通过大量的“红绿灯”**数据**不断地训练让它获得“红灯停绿灯行，黄灯请你等一等”这一**模型**，从而使得行进过程中当图像传感器获得红灯图像时，它就会**推理**出需要启动制动系统来刹车。



2.2 一眼看不出来的$\hat{Y} = kX$

第一天的西瓜卖的不好，于是王婆第二天准备卖香蕉。由于香蕉供应商会看人下菜碟，他给不同的人报价不同。她去询问张三、李四、王五等人后得知“今早张三进货了100斤香蕉共花费110元，李四进货了200斤香蕉共花费180元，王五进货了300斤香蕉共花费300元等等”，如果王婆明天准备进货150斤香蕉，那么她需要准备多少的本钱来进货呢？一眼就能看出来。。。不对，看不出来了！同样的假设用$\hat{Y}$表示预测本钱，X表示香蕉的斤数，下面图中的黑点表示从市场上打听到的多家商户的本钱和进价，我们还是能比较直观的看出来$\hat{Y} = kX$是本钱和斤数的规律，即这是一个一元线性回归。那么我们怎么知道k等于多少是最能表现真实关系的呢？

要真实的表现关系，那么预测本钱$\hat{Y}$和真实Y一定是最接近的才可以，然而商户众多，我们预测的$\hat{Y}_i$（包含多家商户的预测值：$\hat{Y}_1$、$\hat{Y}_2$、$\hat{Y}_3$等等）要如何保证能够更好地反应出真实的Y_i（包含多家商户真实的值：Y_1、Y_2、Y_3等等）?最直接的方法就是让所有的预测与真实的差值的绝对值之和最小，也就是说$\sum_i^n |(\hat{Y}_i - Y_i)|$最小。为了计算方便我们通常使用平方来代替绝对值，也就是$\sum_i^n (\hat{Y}_i - Y_i)^2$最小。通过让这个函数取最小值就能实现我们的任务目标，这个函数我们就叫做**损失函数**。函数取最小值的方法很明显就是让一阶导等于0来计算就行了，计算过程不用管，扔给计算机就行了。



这整个过程就是通过分析来选择要使用的模型（从历史数据中分析出是一元线性回归问题，假设$\hat{Y} = kX$），然后确定优化的算法（找到合适的损失函数并经过计算得到模型中的参数的值，这里是k）。从上诉两个案例中我们了解到了数据、模型、推理整体框架以及模型的设计与参数的求解，万变不离其宗，无论多么复杂的问题都基本是这样的范式。如果王婆同时售卖西瓜、香蕉、橘子等多种水果，

那么本钱和斤数的规律问题就是一个多维线性回归的问题了，需要用到矩阵运算相关公式，这里不做说明，只需要了解它们底层逻辑基本一致即可。

3. 稍微复杂一点的模型（传统机器学习）

3.1 好瓜与坏瓜（二分类）

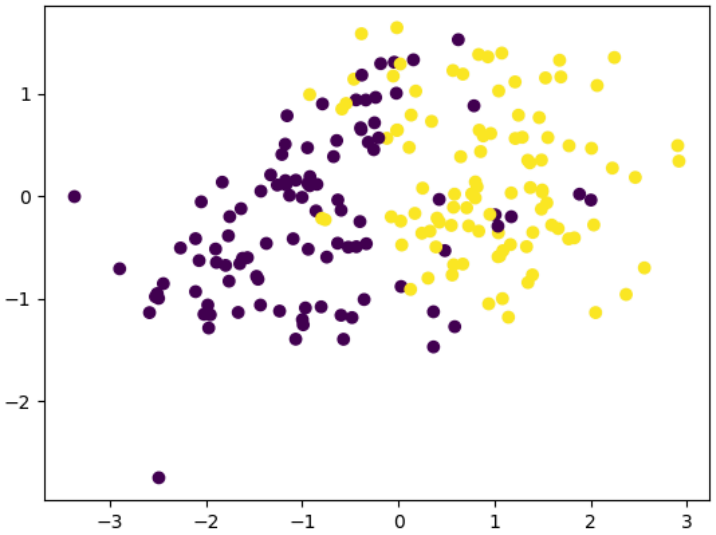
王婆在第三天的时候收到了关于卖出去西瓜的一些反馈，她发现之前进货的西瓜有一部分是好瓜，另一部分是坏瓜。假设进货商每批西瓜出库都会有机器自动检测一些数据，这些数据记录了西瓜的一些**特征**（例如长轴、短轴、色泽、敲击响度等等）与基准之间的偏离值，我们用`sklearn.datasets`库中的`make_classification`函数随机生成一些数据来模拟这些特征，其中X的值的每一行都是由四个特征组成的**特征向量**，y的值用1/0来表示好瓜坏瓜。

```
#生成数据
X, y = make_classification(n_samples=200, n_features=4, random_state=42)
```

```
#输出X的值
[[ 1.68976749e+00 -1.40824123e+00 -9.62936022e-01  1.16349130e+00]
 [ 8.55819924e-01 -4.30645568e-01 -3.90658621e-01  4.35910550e-01]
 ...
 [-8.01477919e-01  1.63636384e+00  7.89288483e-01 -1.07743256e+00]
 [ 1.14492953e+00  2.66995092e+00  5.92077414e-01 -1.17852376e+00]]
```

为了便于直观感受，我们从四个维度中任意选择两个维度的特征来做二维图，其中横纵轴表示选取的两个特征维度，不同的颜色表示不同的类别。

```
#做散点图
plt.scatter(
    X.T[0], #横坐标
    X.T[3], #纵坐标
    c=y
)
plt.show()
```

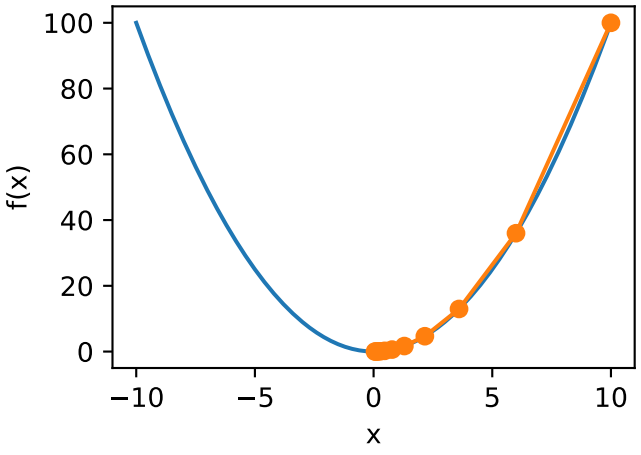


接下来展示逻辑斯特回归算法对该问题进行二分类的全流程，首先对生成的特征数据进行数据集划分（一般我们需要准备训练数据集和测试数据集，首先用训练数据集训练模型，从而确定模型**超参数**的值如上一章$\hat{Y} = kX$中的k，再用测试集预测并且评估该训练好的模型的性能如准确率等**评估指标**），随后利用**梯度下降**等优化算法来训练模型（类似于上一章的求导，因为高维下不好求极值，使用梯度下降等迭代算法可以快速优化得到局部最优解），然后利用评估指标准确率来评估性能，如果性能较好的话我们就可以部署该模型进行新样本的预测了。下面是完整代码。

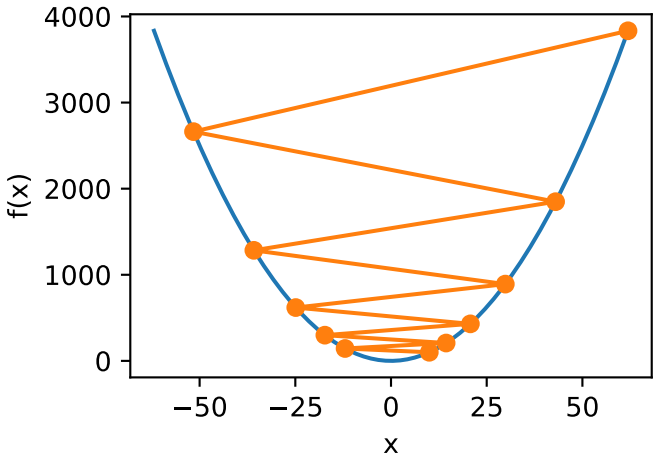
```python
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# 生成模拟数据
X, y = make_classification(n_samples=200, n_features=4, random_state=42)

# 数据集划分
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 训练逻辑斯特回归模型
model = LogisticRegression()
model.fit(X_train, y_train)

# 预测并计算准确率
y_pred = model.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
# 输出准确率
Accuracy: 0.9333333333333333
```

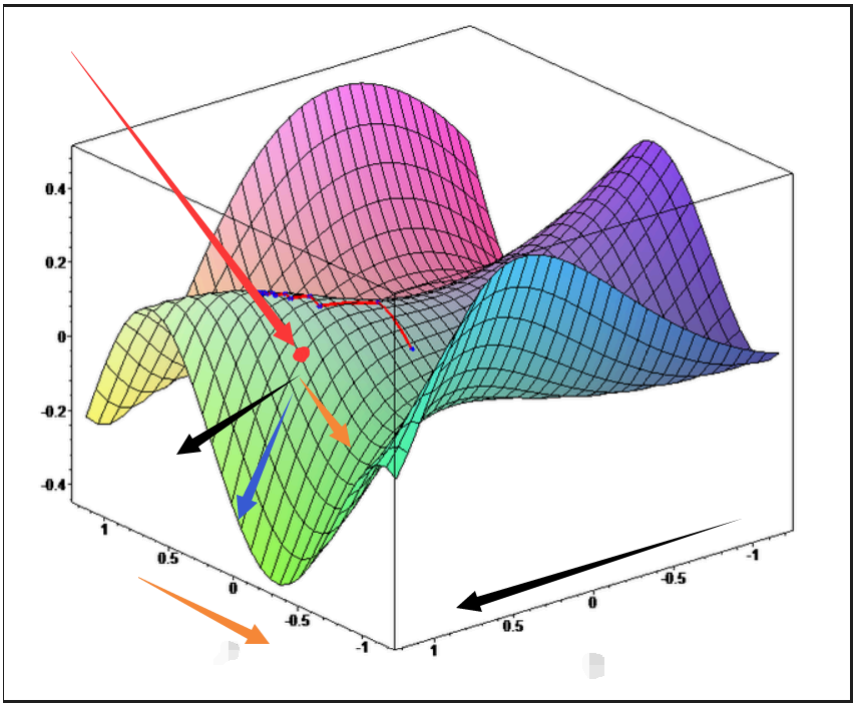
梯度下降算法为什么能找到极值点呢？首先我们看下面这个二维图像，假设我们选取的函数$f(x) = x^2$。在梯度下降中，我们通常先选取一个初始值$x_0 = 10$和**学习率**常数$\eta(\eta > 0)$，每次迭代变化的x值和学习率、一阶导有关，然后通过不断迭代x，直到达到停止条件（因迭代变化的$f(x)$已经为0或者迭代次数完成）。从图中轨迹我们能看出来随着迭代次数的增加，最终确认了极小值在$x = 0$处取得。



不过需要注意的是，如果使用过大的学习率可能导致结果不收敛。当设学习率$\eta = 1.1$时，可以看到x不断越过最优解$x = 0$并逐渐发散。



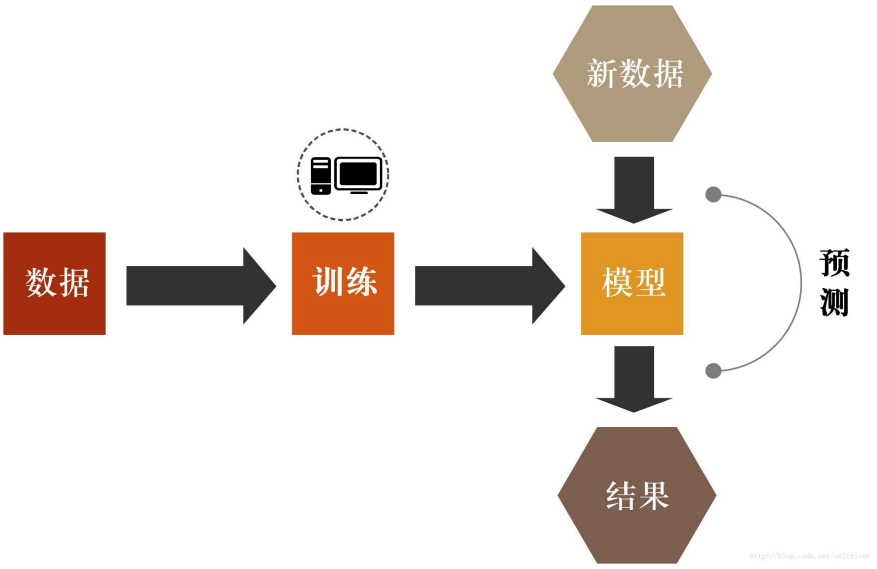
然后我们推广到下面这个三维图像（更高的纬度，三维人也想象不出来），我们的目标就是在这种高维空间找到极值点，这个时候梯度是一个由各个维度的偏导数所组成的一个**「向量」**（如 $\left(\frac{\partial y}{\partial x1}, \frac{\partial y}{\partial x2}\right)$）。除此之外需要了解到如果函数有多个极值点的话，我们可能会求得局部最小值而不是全局最小值。



3.2 机器学习工作流

机器学习工作流（WorkFlow）包含数据**预处理**（Processing）、**模型学习**（Learning）、**模型评估**（Evaluation）、**新样本预测**（Prediction）几个步骤，和上一章的内容基本吻合。

- **数据预处理**：输入（未处理的数据 + 标签）→处理过程（特征处理+幅度缩放、特征选择、维度约减、采样）→输出（测试集 + 训练集）。
- **模型学习**：模型选择、交叉验证、结果评估、超参选择。
- **模型评估**：了解模型对于数据集测试的得分，训练好的模型对于陌生数据的性能越好，它的**泛化**能力越好。
- **新样本预测**：部署好模型来预测新样本。

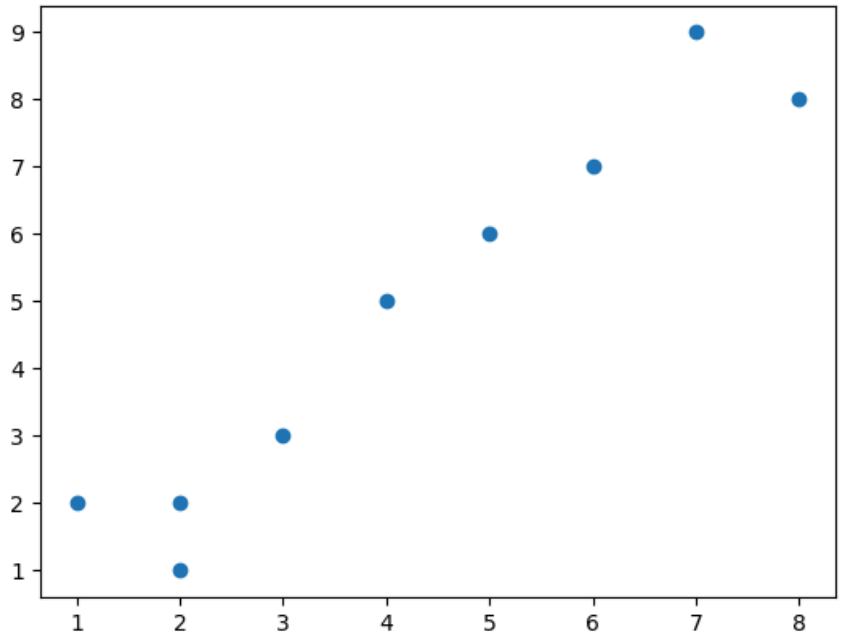


3.3 吃瓜客户营销（聚类）

上诉案例已经涉及回归与分类问题，接下来介绍聚类问题。王婆的摊位最近生意逐渐火爆起来了，她准备对销售记录进行了统计和分析。目前记录的有客户姓名、客户消费额、客户到店次数，她要如何精准把握客群，分别展开进行营销活动呢？为方便计算仅选取很少量的数据（实际情况中采集到的会是很大量的数据），假设统计到的数据如下表所示。

客户	消费额（百元）	到店次数
A	1	2
B	2	1
C	4	5
D	5	6
E	8	8
F	6	7
G	7	9
H	2	2
I	3	3

对该数据做散点图如下，横轴表示消费额，纵轴表示到店次数：



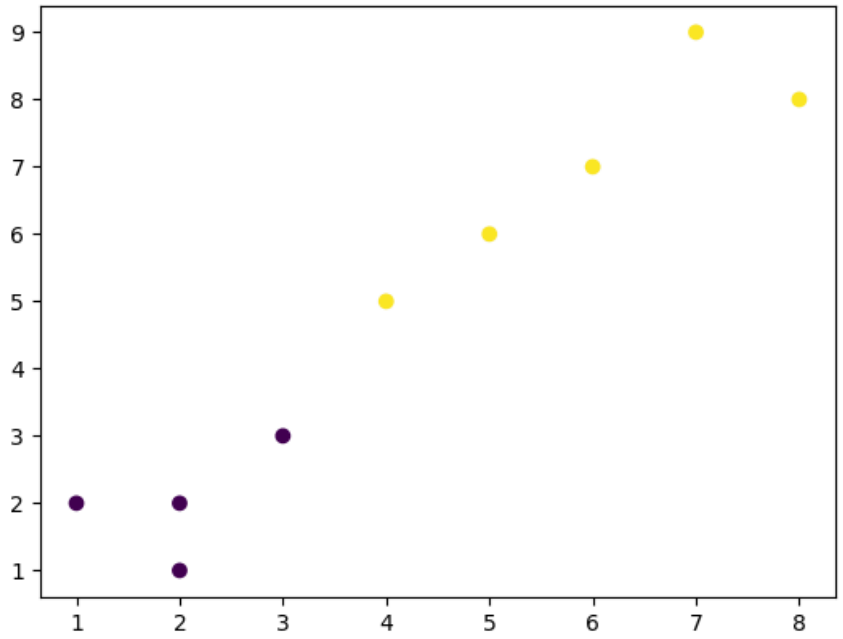
对于这个问题我们发现有些客群属于高粘性客户，他们到店次数和消费额都比较多，另一部分属于低粘性客户，因此可以将他们划分为两个群体分别采取措施，此时可以采取k**均值聚类**（$K-means$）的算法来对数据进行聚类。$K-means$ 的目标是将数据集划分为K个簇（clusters），使得每个数据点属于距离最近的簇中心的那个簇。通过反复迭代，反复调整簇中心的位置和各个点归属于某个簇的分配情况，使得簇内数据点与质心的距离尽可能小，逐步收敛从而获得尽量紧凑、彼此分离的簇。整个流程如下：

1. 随机选择K个簇心（初始中心点）。
2. 每个数据点分配到距离最近的簇心所属的簇。
3. 分配完成后重新计算每个簇的簇心（质心）。
4. 重复步骤 2 和 3，直到簇心不再变化（或达到指定的迭代次数）。

假设我们初始选取H（2，2）和F（6，7）作为两个初始中心点$C1$、$C2$，我们可以得到假设用0表示H簇，用1表示F簇，那么各个客户所属簇的数组可以表示为$[0,0,1,1,1,1,1,0,0]$，其中每个簇的点包含情况为：

```
c1:[array([1, 2]), array([2, 1]), array([2, 2]), array([3, 3])]
c2:[array([4, 5]), array([5, 6]), array([8, 8]), array([6, 7]), array([7, 9])]
```

新的簇心变成了$C1(2,2)$，$C2(6,7)$，此时发现与原簇心一致，因此结束迭代，每个簇的点也求解出来了。到此该问题就得到了求解，聚类完成后的可视化散点图如下所示。

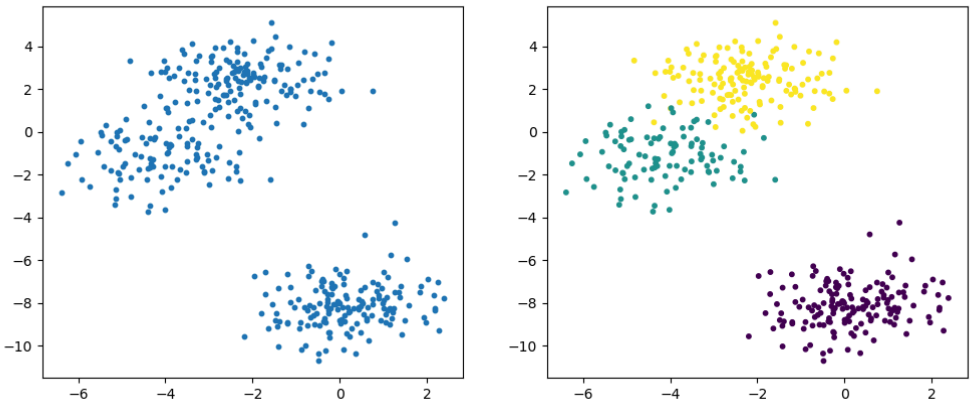


完整的模拟代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import distance

ori_data = np.array([[1,2,4,5,8,6,7,2,3],[2,1,5,6,8,7,9,2,3]])
P=ori_data.T
X=ori_data[0,:]
Y=ori_data[1,:]
# #二维坐标可视化
# plt.scatter(X, Y, s=2)
# plt.show()
C1=[2,2]
C2=[6,7]
cluster1 = np.empty((8,2))
cluster2 = np.empty((8,2))
i = 0
j = 0
#求解
for p in P:
#计算各个点与簇心的距离
    d1=distance.euclidean(p,C1)
    d2=distance.euclidean(p,C2)
#计算各个点的归属
    if(d1>=d2):
        cluster2[i,:]=p
        i=i+1
    else:
        cluster1[j,:]=p
        j=j+1
cluster1=cluster1[:j,:]
cluster2=cluster2[:i,:]
#求解新簇心
```

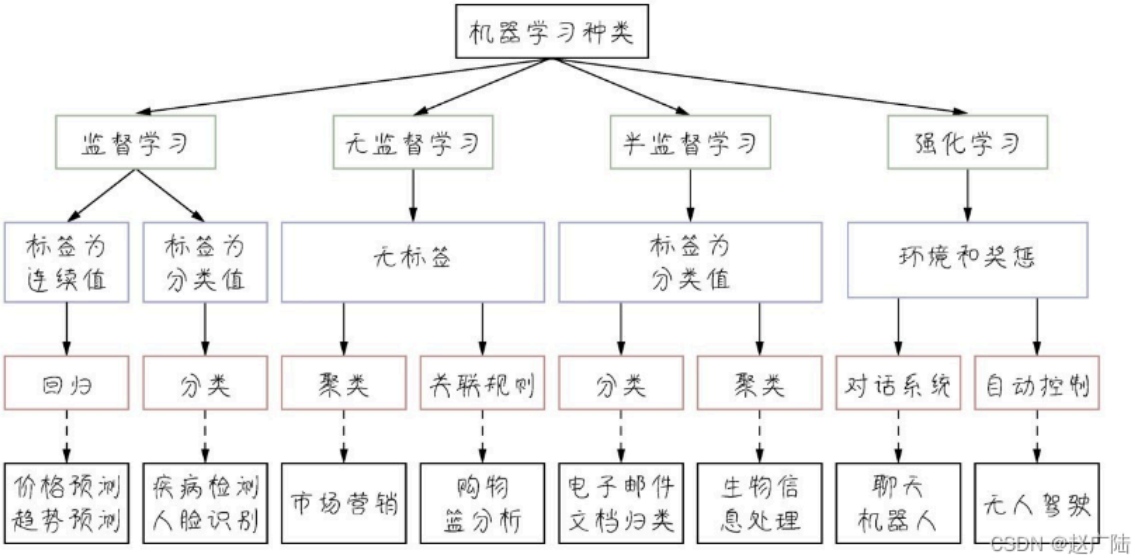
```
centroid1 = np.mean(cluster1, axis=0)
centroid2 = np.mean(cluster2, axis=0)
```

除此之外，可以直接使用`sklearn.cluster`库中的`KMeans`聚类函数来实现聚类。为了方便更直观的观察结果，接下来使用三个簇分别$[155, 100, 145]$共四百个数据来模拟聚类前后效果图。



3.4 机器学习算法主要类别

机器学习算法最主要的类别有：监督学习、无监督学习和强化学习等等，他们都满足工作流：从给定的训练数据集中学习出一个函数，当新的数据到来时，可以根据这个函数预测结果。**监督学习**：监督学习的训练集要求是包括输入和输出，也可以说是特征和目标（标签）。训练集中的目标是由人为标注的（我们会告诉模型这是好瓜或者坏瓜）。常见的监督学习算法包括回归分析和统计分类。**无监督学习**：与监督学习相比，训练集没有人为标注的结果（模型不知道这是好瓜坏瓜）。常见的无监督学习算法有聚类和降维。**强化学习**：通过观察来学习做成如何的动作。每个动作都会对环境有所影响，学习对象根据观察到的周围环境的反馈来做出判断（例如下棋的时候目前这个残局这颗子下这个位置是正收益或是负收益，正收益就奖励，负收益就惩罚）。



4. 更复杂一点的模型（深度学习）

待更新。。。。。