



浙江大学数学科学学院

---

## 前沿数学专题讨论

---

### 课程讲义

*Submitted To:*

张南松老师

22-23 秋冬学期

*Submitted By :*

吴凡

农业工程 + 统计学

# Contents

<b>1 决策树</b>	<b>4</b>
1.1 问题背景 . . . . .	4
1.2 基本算法 . . . . .	6
1.3 ID3 算法模型 . . . . .	6
1.4 C4.5 算法模型 . . . . .	8
1.5 CART 决策树模型 . . . . .	9
1.6 决策树的剪枝 . . . . .	11
1.7 算法实例 . . . . .	12
<b>2 支持向量机</b>	<b>16</b>
2.1 概述 . . . . .	16
2.1.1 简介 . . . . .	16
2.1.2 为何叫支持向量机 . . . . .	16
2.1.3 直观理解 . . . . .	16
2.2 线性可分 SVM——hard margin . . . . .	18
2.2.1 超平面与间隔 . . . . .	19
2.2.2 数学模型 . . . . .	19
2.3 算法实例 . . . . .	22
2.3.1 鸢尾花分类问题背景 . . . . .	22
2.3.2 SVM 分类 . . . . .	22
<b>3 最大期望算法</b>	<b>24</b>
3.1 概述 . . . . .	24

3.1.1 简介 . . . . .	24
3.1.2 算法思想 . . . . .	24
3.1.3 EM 和 MLE . . . . .	26
3.2 数学模型 . . . . .	27
3.3 EM 的应用 . . . . .	29
3.3.1 EM 的优缺点 . . . . .	29
3.3.2 EM 的应用 . . . . .	29
<b>4 套索算法</b>	<b>31</b>
4.1 背景 . . . . .	31
4.1.1 线性模型的最小二乘估计 . . . . .	31
4.1.2 岭估计的提出 . . . . .	32
4.2 LASSO . . . . .	33
4.2.1 Lasso 的优势 . . . . .	34
4.2.2 Lasso 最优解 . . . . .	35
4.2.3 ElasticNet 模型 . . . . .	36
4.3 算法实例 . . . . .	37
4.3.1 使用 R 进行 Ridge 回归 . . . . .	37
4.3.2 使用 R 进行 Lasso 回归 . . . . .	39
4.3.3 ElasticNet 模型 . . . . .	40
<b>5 自然进化策略算法</b>	<b>42</b>
5.1 Evolution Strategy . . . . .	42
5.2 CMA Evolution Strategy . . . . .	45
5.2.1 Updating the Mean . . . . .	45
5.2.2 Controlling the Step Size . . . . .	47
5.2.3 Adapting the Covariance Matrix . . . . .	49
<b>6 深度强化学习</b>	<b>52</b>
6.1 前言 . . . . .	52
6.1.1 PG . . . . .	53
6.1.2 DPG . . . . .	54

6.1.3 DDPG . . . . .	54
6.2 DDPG 算法 . . . . .	55
6.2.1 算法背景 . . . . .	55
6.2.2 算法相关概念和定义 . . . . .	56
6.2.3 DDPG 实现框架和算法 . . . . .	58
6.3 应用 . . . . .	59
<b>7 总结和体会</b>	<b>63</b>
7.1 三种决策树算法之比较 . . . . .	63
7.2 弹性网模型 . . . . .	68
7.3 心得与总结 . . . . .	71
<b>8 参考文献</b>	<b>73</b>
<b>9 附录</b>	<b>76</b>
9.1 附录 1 决策树代码 . . . . .	76
9.2 附录 2 SVM 代码 . . . . .	78
9.3 附录 3 EM 代码 . . . . .	81
9.4 附录 4 Lasso 代码 . . . . .	82
9.5 附录 5 ES 代码 . . . . .	83
9.6 附录 6 CMA-ES 代码 . . . . .	86
9.7 附录 7 DDPG 代码 . . . . .	89

# Chapter 1

## 决策树

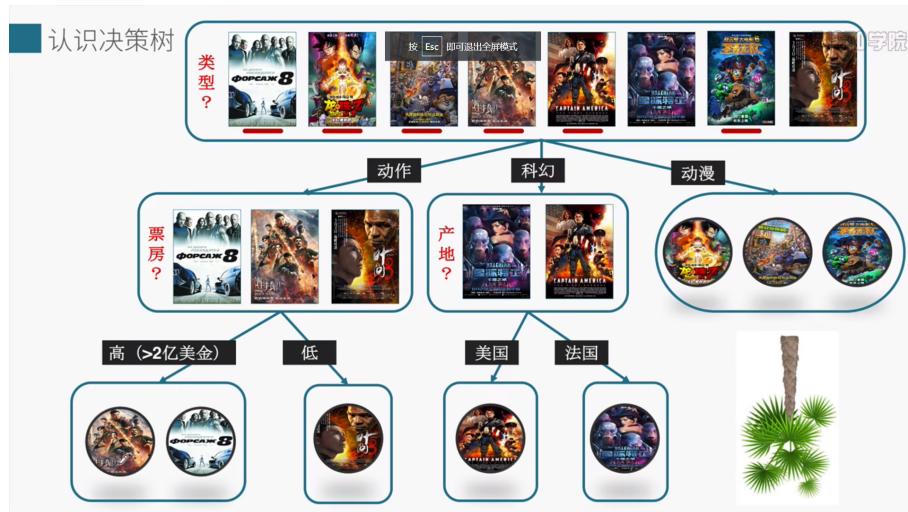
### 1.1 问题背景

小明看过以下 6 部电影, 请你根据这 6 部电影的特点来推断小明是否喜欢看某部电影。

小明观影记录

序号	片名
1	疯狂动物城
2	美国队长2
3	龙珠Z：复活的弗利萨
4	速度与激情8
5	战狼II
6	赛尔号大电影6：圣者无敌

为建立决策树模型, 引入 2 部小明没看过的电影作为对照。



决策树模型的任务：从给定的训练数据集学得一个模型对新示例进行分类。可看作对“当前示例属于这类吗？”这个问题的“决策”或“判定”过程。

这个决策的过程通过决策树来完成。决策过程中提出的每个判定问题都是对某个属性的“测试”。

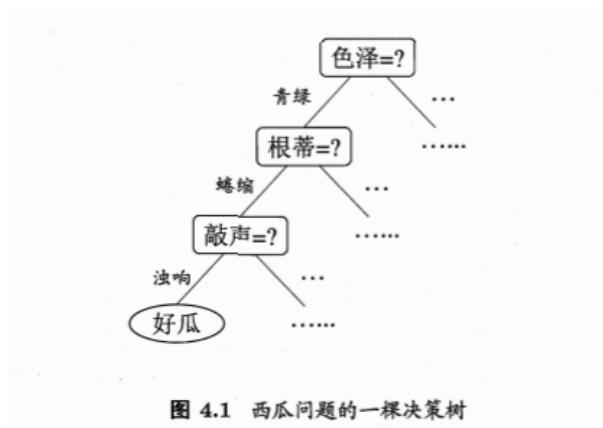


图 4.1 西瓜问题的一棵决策树

决策树学习的目的：产生一棵泛化能力强，即处理未见示例能力强的决策树。

## 1.2 基本算法

决策树模型基本流程遵循简单且直观的“分而治之”策略。

算法流程如下：

---

```

输入: 训练集  $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$ ;
       属性集  $A = \{a_1, a_2, \dots, a_d\}$ .
过程: 函数 TreeGenerate( $D, A$ )
1: 生成结点 node;
2: if  $D$  中样本全属于同一类别  $C$  then
3:   将 node 标记为  $C$  类叶结点; return
4: end if
5: if  $A = \emptyset$  OR  $D$  中样本在  $A$  上取值相同 then
6:   将 node 标记为叶结点, 其类别标记为  $D$  中样本数最多的类; return
7: end if
8: 从  $A$  中选择最优划分属性  $a_*$ ;
9: for  $a_*$  的每一个值  $a_*^v$  do
10:   为 node 生成一个分支; 令  $D_v$  表示  $D$  中在  $a_*$  上取值为  $a_*^v$  的样本子集;
11:   if  $D_v$  为空 then
12:     将分支结点标记为叶结点, 其类别标记为  $D$  中样本最多的类; return
13:   else
14:     以 TreeGenerate( $D_v, A \setminus \{a_*\}$ ) 为分支结点
15:   end if
16: end for
输出: 以 node 为根结点的一棵决策树

```

---

## 1.3 ID3 算法模型

一般而言，随着划分过程不断进行，我们希望决策树的分支结点所包含的样本尽可能属于同一类别，即结点的“纯度”(purity)越来越高。

“信息熵”为度量样本集合纯度常用的一种指标，假定当前样本集合  $D$  中第  $k$  类样本所占的比例为  $p_k (k = 1, 2, \dots, |y|)$ ，则  $D$  的信息熵定义为

$$Ent(D) = - \sum_{k=1}^{|y|} p_k \log_2 p_k \quad (1.1)$$

$Ent(D)$  的值越小，则  $D$  的纯度越高。

假定离散属性  $a$  有  $V$  个可能的取值  $\{a^1, a^2, \dots, a^V\}$ , 若使用  $a$  来对样本集  $D$  进行划分, 则会产生  $V$  个分支结点, 其中第  $v$  个分支结点包含了  $D$  中所有在属性  $a$  上取值为  $a^v$  的样本, 记为  $D^v$ , 可计算出  $Ent_a(D)$ , 于是可计算出用属性  $a$  对样本集  $D$  进行划分所获得的”信息增益”

$$Gain(D, a) = Ent(D) - Ent_a(D) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{D} Ent(D^v) \quad (1.2)$$

一般而言, 信息增益越大, 则意味着使用属性  $a$  来进行划分所获得的”纯度提升”越大, 因此可选择属性  $a_* = argmax_{a \in A} Gain(D, a)$

举例:

小明观影记录					
序号	片名	类型	产地	票房	是否观影
1	龙珠Z:复活的弗利萨	动漫	日本	低1,100万	1
2	美国队长3	科幻	美国	低124,637万	1
3	疯狂动物城	动漫	美国	低153,035万	1
4	速度与激情8	动作	美国	高267,098万	1
5	战狼2	动作	中国	高548,461万	1
6	赛尔号大电影6 : 圣者无敌	动漫	中国	低9758.2万	1
7	星级特工	科幻	法国	低32,047万	0
8	叶问3	动作	中国	低77,031万	0

$$Ent(D) = -(\frac{6}{8} * ln\frac{6}{8} + \frac{2}{8} * ln\frac{2}{8}) = 0.8112$$

$$Ent(D) = \frac{3}{8} * (-\frac{2}{3} * ln\frac{3}{3} - \frac{1}{3} * ln\frac{1}{3}) + \frac{2}{8} * (-\frac{1}{2} * ln\frac{1}{2} - \frac{1}{2} * ln\frac{1}{2}) + \frac{3}{8} * (-\frac{2}{3} * ln\frac{2}{3} - \frac{1}{3} * ln\frac{1}{3}) = 0.5944$$

$$\text{因此, } Gain(D, \text{产地}) = Ent(D) - Ent(D) = 0.2168$$

$$\text{同理可求得, } Ent(D) = 0.3444 \quad Ent(D) = 0.6887$$

$$Gain(D, \text{类型}) = Ent(D) - Ent(D) = 0.4668$$

$$Gain(D, \text{票房}) = Ent(D) - Ent(D) = 0.1225$$

因此,  $Gain(D, \text{产地}) > Gain(D, \text{类型}) > Gain(D, \text{票房})$

属性”产地”的信息增益最大, 于是它被选为划分属性, 随后, 决策树学习算法对每个分支结点做进一步划分。



上述过程为决策树算法中最常用的算法模型——ID3 算法模型。

## 1.4 C4.5 算法模型

按照信息增益准则， $Gain(D, a) = Ent(D) - \sum_{v=1}^V \frac{|D^v|}{|D|} Ent(D^v)$ ，对可取值数目较多的属性有所偏好，如在“小明观影”的例子中，若把“序号”作为划分属性，此时， $Ent(D) = 6 * \frac{1}{8} * (-\frac{1}{1} * ln \frac{1}{1}) + 2 * \frac{1}{8} * (-\frac{0}{1} * ln \frac{0}{1}) = 0$ ，则信息增益非常大，而，若模型以“序号”作为划分属性，显然，此时这些分支结点的纯度已达最大，然而，这样的决策树显然不具有泛化能力，无法对新样本进行有效预测。

C4.5 算法模型不直接使用信息增益，而是使用“增益率”(gain ratio) 来选择最优划分属性。

$$Gain_{ratio}(D, a) = \frac{Gain(D, a)}{IV(a)} \quad (1.3)$$

其中， $IV(a) = -\sum_{v=1}^V \frac{|D^v|}{|D|} \log_2 \frac{|D^v|}{|D|}$  为属性 a 的“固有值”，属性 a 的可能取值数目越多，则 IV(a) 的值通常会越大。

增益率准则对可取值数目较少的属性有所偏好，因此 C4.5 算法并不是直接选

择增益率最大的候选划分属性，而是使用了一个启发式：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

## 1.5 CART 决策树模型

CART 决策树模型采用“基尼系数”(Gini index) 来选择划分属性

$$Gini(D) = 1 - \sum_{k=1}^{|y|} p_k^2 \quad (1.4)$$

直观来说， $Gini(D)$  反映了从数据集  $D$  中随机抽取两个样本，其类别标记不一致的概率， $Gini(D)$  越小，则数据集  $D$  的纯度越高。

**Gini计算示例**

$$GINI = 1 - \sum_{i=1}^c p(i)^2$$

C1	0	$P(C1) = 0/6 = 0$	$P(C2) = 6/6 = 1$
C2	6	$Gini = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$	

C1	1	$P(C1) = 1/6$	$P(C2) = 5/6$
C2	5	$Gini = 1 - (1/6)^2 - (5/6)^2 = 0.278$	

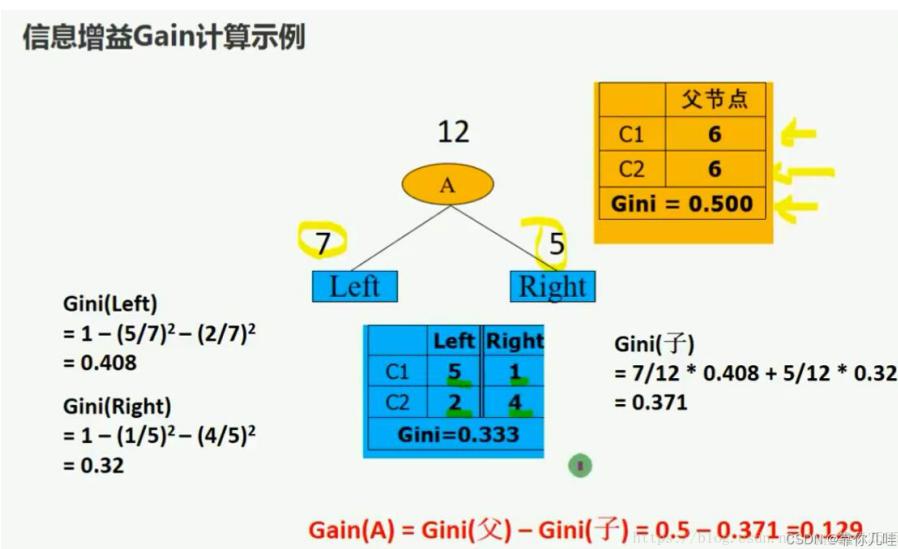
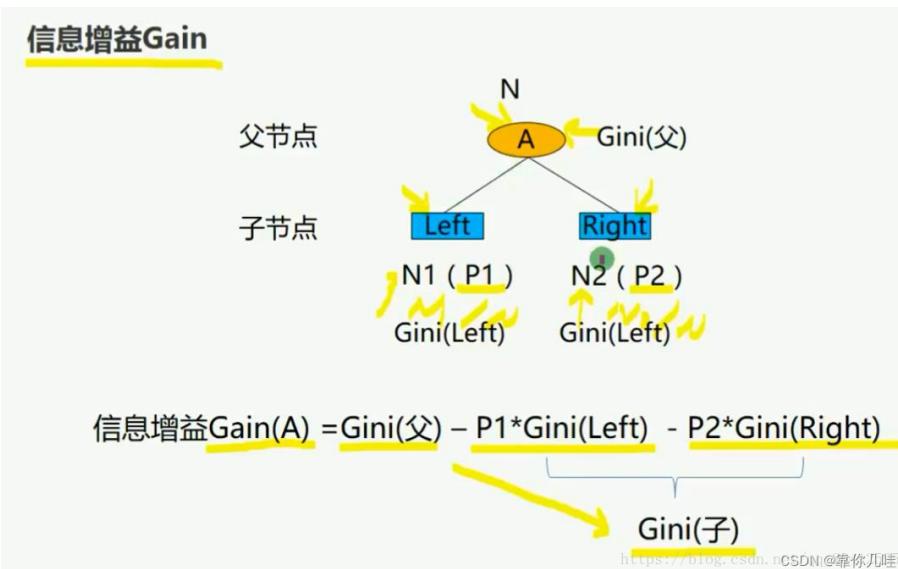
C1	2	$P(C1) = 2/6$	$P(C2) = 4/6$
C2	4	$Gini = 1 - (2/6)^2 - (4/6)^2 = 0.444$	

https://blog.csdn.net/CSDNi@靠你几壁

属性  $a$  的基尼指数定义为：

$$Gini_{index}(D, a) = \sum_{v=1}^V \frac{|D^v|}{|D|} Gini(D^v) \quad (1.5)$$

于是，选择使得划分后基尼指数最小的属性作为最优划分属性，即  $a_* = argmin Gini_{index}(D, a)$



## 1.6 决策树的剪枝

决策树生成算法递归地产生决策树，直到不能继续下去为止，这样产生的树往往会出现过拟合现象。产生过拟合的原因在于学习时过多地考虑如何提高对训练数据的正确分类，从而构建出过于复杂的决策树。解决这个问题的办法是对已生成的决策树进行简化。

在决策树学习中将已生成的树进行简化的过程称为剪枝。具体是从已生成的树上裁掉一些子树或叶节点，并将其根节点或父节点作为新的叶节点，从而简化分类树模型。

剪枝策略分为预剪枝 (prepruning) 和后剪枝 (post-pruning)。

预剪枝：在决策树生成过程中，对每个节点在划分前先进行估计，若当前节点的划分不能带来决策树泛化性能提升，则停止划分并将当前节点标记为叶节点。

后剪枝：先从训练集中生成一棵完整的决策树，然后自底向上地对非叶节点进行考察，若将该节点对应的子树替换成叶节点能带来决策树泛化能力的提升，则将该子树替换成叶节点。

## 1.7 算法实例

使用银行营销数据集,这些数据共包括 16 个特征,输出标签有 2 个(“是”、“否”),即客户是否已订阅定期存款。我们将采用其中的三个特征 “年龄”、“年平均余额”以及 “该月与该客户最后一次联系的日期” 对客户是否订阅定期存款来进行预测。

### 属性信息

- age 年龄 (数字)
- job 职务: 职务类型 (类别: “管理员”, “蓝领”, “企业家”, “女佣”, “管理”, “退休”, “自雇”, “服务”, “学生”, “技术员”, “待业”, “未知”)
- marital 婚姻: 婚姻状况 (类别: “已婚”, “离婚”, “单身”, “未知”)
- education 教育 (类别: “未知”, “中学”, “小学”, “高等教育”)
- default 违约: 信用违约吗? (类别: “否”, “是”)
- balance 收支: 年平均余额
- housing 住房: 有住房贷款吗? (分类: “否”, “是”)
- loan 贷款: 有个人贷款吗? (类别: “否”, “是”)

等等

输出类别信息: 客户是否已定阅?

```
import pandas as pd

bm = pd.read_csv('./bank/bank.txt', sep=';')

X = bm[['age', 'balance', 'day']].values[:200, :]
y = bm['y'].values[:200]

bm.head()
```

将数据集划为 70% 作为训练集, 30% 划分为测试集。

	age	job	marital	education	default	balance
0	30	unemployed	married	primary	no	1787
1	33	services	married	secondary	no	4789
2	35	management	single	tertiary	no	1350
3	30	management	married	tertiary	no	1476
4	59	blue-collar	married	secondary	no	0

5 rows × 7 columns

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y)
```

使用 scikit-learn 训练一个决策树

```
from sklearn.tree import DecisionTreeClassifier

# 采用gini系数进行度量，树的最大深度设置为4
tree = DecisionTreeClassifier(criterion='gini',
                               max_depth=4,
                               random_state=1)

# 训练
tree.fit(X_train, y_train)
```

DecisionTreeClassifier() 函数参数:

```
class sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best',
                                         max_depth=None,
                                         min_samples_split=2,
                                         min_samples_leaf=1,
                                         min_weight_fraction_leaf=0.0,
                                         max_features=None, random_state=None,
                                         max_leaf_nodes=None,
                                         min_impurity_decrease=0.0,
                                         min_impurity_split=None,
                                         class_weight=None, presort=False)
```

criterion: 默认 gini, 即 CART 算法。

max depth: 决策树最大深度，默认值是'None'. 一般数据比较少或者特征少的时候可以不用管这个值，如果模型样本数量多，特征也多时，推荐限制这个最大深度，具体取值取决于数据的分布。常用的可以取值 10 到 100 之间，常用来解决过拟合。

random state: 是用来设置决策树分枝中随机模式的参数，在高维度时 sklearn 决策树的特征随机性会很明显，低维度的数据（比如鸢尾花数据集），随机性几乎不会显现。高维数据下我们设置 random state 并配合 splitter 参数可以让模型稳定下来，保证同一数据集下是决策树结果可以多次复现，便于模型参数优化。

计算模型准确率：

```
from sklearn.metrics import accuracy_score
y_train_pre = tree.predict(X_train)
y_test_pre = tree.predict(X_test)
print("training accuracy: %.3f" % accuracy_score(y_train_pre, y_train))
print("testing accuracy: %.3f" % accuracy_score(y_test_pre, y_test))
```

Output:

training accuracy: 0.907

testing accuracy: 0.817

使用 Graphviz 库进行对决策树进行可视化：

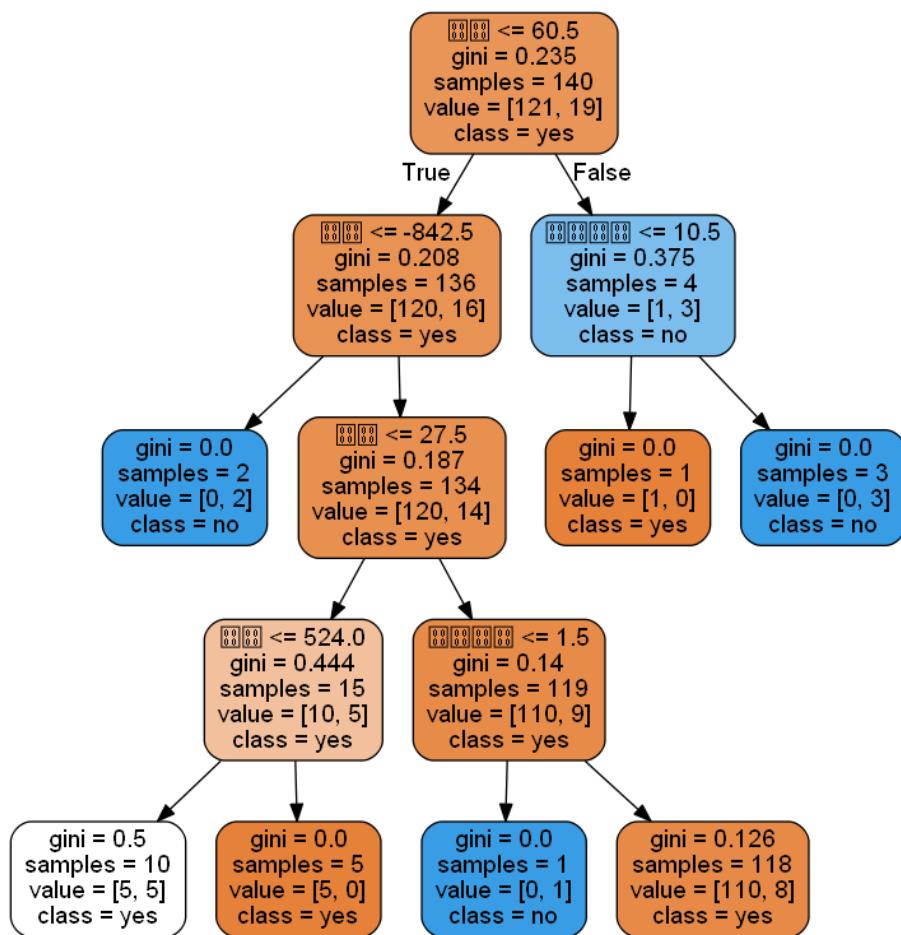
```
from pydotplus import graph_from_dot_data
from sklearn.tree import export_graphviz

# 生成 .dot 文件
dot_data = export_graphviz(tree,
                           filled=True,
                           rounded=True,
                           class_names=['yes',
                                         'no'],
                           feature_names=['年龄',
                                         '收支',
                                         '联系日期'],
                           out_file=None)
```

```
# export_graphviz(tree, out_file='tree.dot', feature_names=['age', 'balance', 'duration'])

# 将.dot文件存为图像文件
graph = graph_from_dot_data(dot_data)

# 将图像文件写入
graph.write_png('tree.png')
```



# Chapter 2

## 支持向量机

### 2.1 概述

#### 2.1.1 简介

SVM，英文全称为 Support Vector Machine，中文名为支持向量机，由数学家 Vapnik 等人早在 1963 年提出。在深度学习兴起之前，SVM 一度风光无限，是机器学习近几十年来最为经典的，也是最受欢迎的分类方法之一。

#### 2.1.2 为何叫支持向量机

SVM 的本质模型特征空间中最大化间隔的**线性分类器**，是一种**二分类模型**。

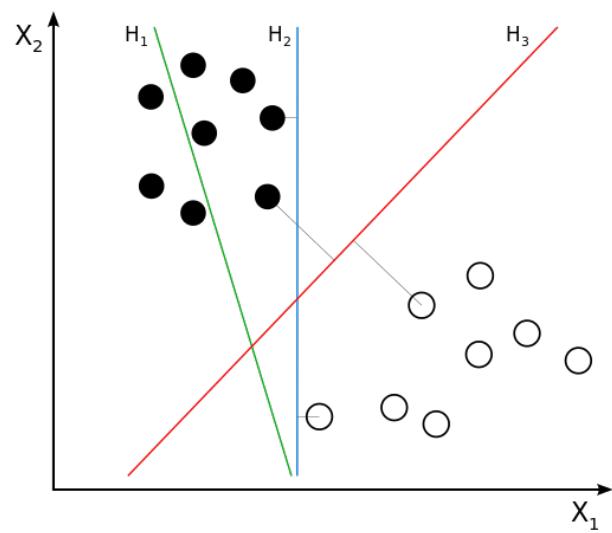
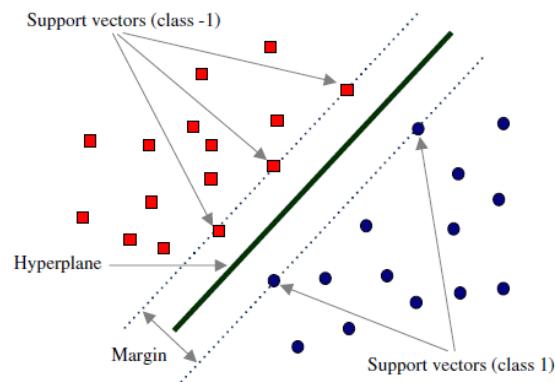
支持向量（Support vector）：离分类超平面（Hyper plane）最近的样本点。

其核心的理念是：支持向量样本会对识别的问题起关键作用

SVM 的学习策略就是间隔最大化。

#### 2.1.3 直观理解

请看下图，图中有分别属于两类的一些二维数据点和三条直线。如果三条直线分别代表三个分类器的话，请问哪一个分类器比较好？



我们凭直观感受应该觉得答案是 H3。首先 H1 不能把类别分开，这个分类器肯定是不行的；H2 可以，但分割线与最近的数据点只有很小的间隔，如果测试数据有一些噪声的话可能就会被 H2 错误分类（即对噪声敏感、泛化能力弱）。H3 以较大间隔将它们分开，这样就能容忍测试数据的一些噪声而正确分类，是一个泛化能力不错的分类器。

对于支持向量机来说，数据点若是  $p$  维向量，我们用  $p-1$  维的超平面来分开这些点。但是可能有许多超平面可以把数据分类。最佳超平面的一个合理选择就是以最大间隔把两个类分开的超平面。因此，SVM 选择能够使离超平面最近的数据点的到超平面距离最大的超平面。

以上介绍的 SVM 只能解决线性可分的问题，为了解决更加复杂的问题，支持向量机学习方法有一些由简至繁的模型：

- **线性可分 SVM:** 当训练数据线性可分时，通过硬间隔 (hard margin, 什么是硬、软间隔下面会讲) 最大化可以学习得到一个线性分类器，即硬间隔 SVM，如上图的的 H3。
- **线性 SVM:** 当训练数据不能线性可分但是可以近似线性可分时，通过软间隔 (soft margin) 最大化也可以学习到一个线性分类器，即软间隔 SVM。
- **非线性 SVM:** 当训练数据线性不可分时，通过使用核技巧 (kernel trick) 和软间隔最大化，可以学习到一个非线性 SVM。

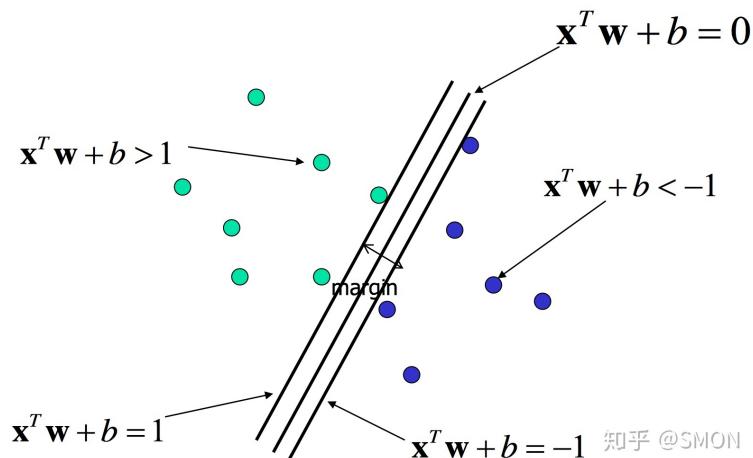
## 2.2 线性可分 SVM——hard margin

考虑如下形式的线性可分的训练数据集： $(X_1, y_1), (X_2, y_2), \dots, (X_N, y_N)$

$y_i \in \{+1, -1\}$ ,  $y_i = +1$  时表示  $X_i$  属于正类别， $y_i = -1$  时表示  $X_i$  属于负类别。

### 2.2.1 超平面与间隔

超平面方程为  $W^T X + b = 0$ , 可以规定法向量指向的一侧为正类, 另一侧为负类。下图画出了三个平行的超平面, 法方向取左上方向。



为了找到最大间隔超平面, 我们可以先选择分离两类数据的两个平行超平面, 使得它们之间的距离尽可能大。在这两个超平面范围内的区域称为“间隔 (margin)”, 最大间隔超平面是位于它们正中间的超平面。这个过程如上图所示。

### 2.2.2 数学模型

判别模型:  $y_i = \text{sign}(W^T X_i + b)$

$$\text{模型判别正确} \Leftrightarrow \begin{cases} W^T X_i + b > 0, y_i = +1 \\ W^T X_i + b < 0, y_i = -1 \end{cases} \Leftrightarrow y_i(W^T X_i + b) > 0, i = 1, 2, \dots, N$$

$$\text{margin}(w, b) = \min_{w, b, X_i} \text{distance}(w, b, X_i) = \min_{w, b, X_i} \frac{|w^T X_i + b|}{\|w\|}$$

SVM 的中心思想: 找到最大间隔超平面, 即使 margin 取到最大值。则优化问题可用如下数学模型表示:

$$\begin{cases} \max_{w,b} \frac{1}{\|w\|} \\ s.t. y_i(w^T X_i + b) \geq 1, i = 1, 2, \dots, N \end{cases} \quad (2.1)$$

$$\Leftrightarrow \begin{cases} \min_{w,b} \frac{1}{2} w^T w \\ s.t. 1 - y_i(w^T X_i + b) \leq 0, i = 1, 2, \dots, N \end{cases} \quad (2.2)$$

问题转化为纯粹的凸优化问题，用拉格朗日乘子法求最优解。

$$L(w, b, \lambda) = \frac{1}{2} w^T w + \sum_{i=1}^N \lambda_i [1 - y_i(w^T X_i + b)] \quad (2.3)$$

$$\Rightarrow \begin{cases} \min_{w,b} \max_{\lambda} L(w, b, \lambda) \\ s.t. \lambda_i \geq 0, i = 1, 2, \dots, N \end{cases} \quad (2.4)$$

引入本拉格朗日方程的对偶方程。我们称上式所述问题为原始问题 (primal problem)，可以应用拉格朗日乘子法构造拉格朗日函数 (Lagrange function) 再通过求解其对偶问题 (dual problem) 得到原始问题的最优解。转换为对偶问题来求解的原因是：

1. 对偶问题更易求解，由下文知对偶问题只需优化一个变量且约束条件更简单；
2. 能更加自然地引入核函数，进而推广到非线性问题。

对偶问题：

$$\begin{cases} \max_{\lambda} \min_{w,b} L(w, b, \lambda) \\ s.t. \lambda_i \geq 0, i = 1, 2, \dots, N \end{cases} \quad (2.5)$$

$$\frac{\partial L}{\partial b} = \frac{\partial \frac{1}{2} w^T w + \sum_{i=1}^N \lambda_i [1 - y_i(w^T X_i + b)]}{\partial b} = - \sum_{i=1}^N \lambda_i y_i = 0 \quad (2.6)$$

$$\Rightarrow \sum_{i=1}^N \lambda_i y_i = 0 \quad (2.7)$$

带入原式  $L(w, b, \lambda)$  中，则

$$L(w, b, \lambda) = \frac{1}{2} w^t w + \sum_{i=1}^N \lambda_i - \sum_{i=1}^N \lambda_i y_i w^T X_i \quad (2.8)$$

$$\frac{\partial L}{\partial w} = \frac{1}{2} w - \sum_{i=1}^N \lambda_i y_i X_i = 0 \quad (2.9)$$

$$\Rightarrow w^* = \sum_{i=1}^N \lambda_i y_i X_i \quad (2.10)$$

将  $w^*$  带入原式  $L(w, b, \lambda)$  中，

$$\begin{aligned} L(w, b, \lambda) &= -\frac{1}{2} (\sum_{i=1}^N \lambda_i y_i X_i)^T (\sum_{j=1}^N \lambda_j y_j X_j) + \sum_{i=1}^N \lambda_i \\ &= -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y_i y_j X_i^T X_j + \sum_{i=1}^N \lambda_i \end{aligned} \quad (2.11)$$

因而，可通过逐项求导，求出  $\lambda_i$  的值。

## 2.3 算法实例

### 2.3.1 鸢尾花分类问题背景

一名植物学爱好者收集了鸢尾花的一些测量数据：花瓣的长度和宽度以及花萼的长度和宽度。他还有一些鸢尾花分类的测量数据，这些花已经被植物学专家鉴定为属于 versicolor、setosa 或 virginica 三个品种之一。

数据集主要属性：

	萼片 长度	萼片 宽度	花瓣 长度	花瓣 宽度	类型
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

每条记录包含 5 项基本信息：花萼的长度、花萼的宽度、花瓣的长度、花瓣的宽度以及鸢尾花的类别。

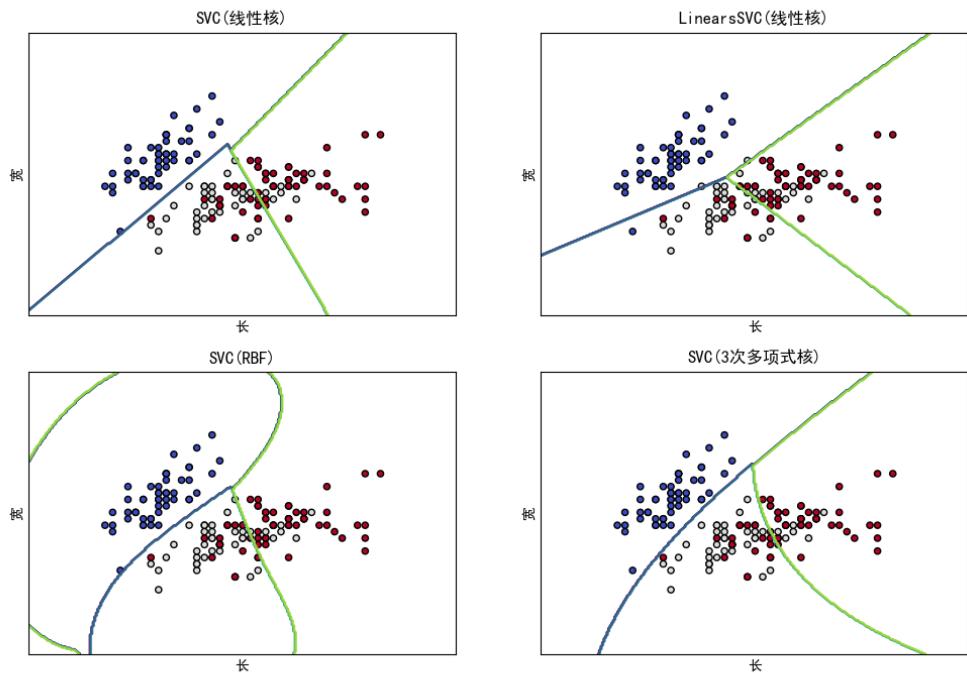
### 2.3.2 SVM 分类

现采用 Python sklearn 中的 SVM 模块对鸢尾花数据集进行分类。

采用的 SVM 分类方法分别为：

- SVC(线性核)
- Linears(线性核)
- SVC(RBF)
- SVC(3 次多项式核)

程序分类可视化结果如下：



# Chapter 3

## 最大期望算法

### 3.1 概述

#### 3.1.1 简介

EM，英文全称为 Expectation-Maximization Algorithm，中文名为最大期望算法，是在概率模型中寻找参数最大似然估计或者最大后验估计的算法，其中概率模型依赖于无法观测的隐性变量。

最大期望算法经过两个步骤交替进行计算：

1. 第一步是计算期望 (E)，利用对隐藏变量的现有估计值，计算其最大似然估计值；
2. 第二步是最大化 (M)，最大化在 E 步上求得的最大似然值来计算参数的值。M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。

#### 3.1.2 算法思想

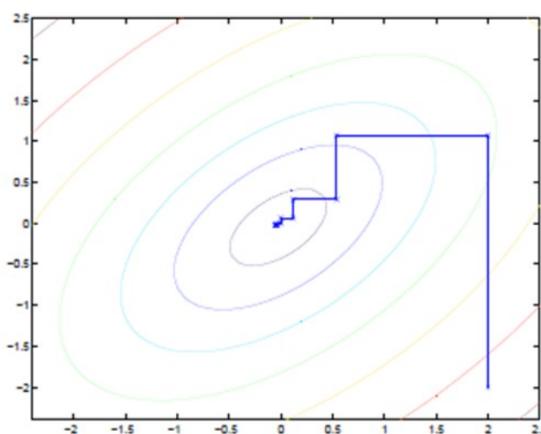
假定你是一五星级酒店的厨师，现在需要把锅里的菜平均分配到两个碟子里。如果只有一个碟子乘菜那就什么都不用说了，但问题是又有 2 个碟子，正因为根本无法估计一个碟子里应该乘多少菜，所以无法一次性把菜完全平均分配。

解法：大厨先把锅里的菜一股脑倒进两个碟子里，然后看看哪个碟子里的菜多，就把这个碟子中的菜往另一个碟子中匀匀，之后重复多次匀匀的过程，直到两个碟子中菜的量大致一样。上面的例子中，平均分配这个结果是“观测数据  $z$ ”，为实现平均分配而给每个盘子分配多少菜是“待求参数”，分配菜的手感就是“概率分布”。

EM 算法的思想：

1. 给 自主规定个初值（既然我不知道想实现“两个碟子平均分配锅里的菜”的话每个碟子需要有多少菜，那我就先估计个值）；
2. 根据给定观测数据和当前的参数，求未观测数据  $z$  的条件概率分布的期望（在上一步中，已经根据手感将菜倒进了两个碟子，然后这一步根据“两个碟子里都有菜”和“当前两个碟子都有多少菜”来判断自己倒菜的手感）；
3. 上一步中  $z$  已经求出来了，于是根据极大似然估计求最优的’（手感已经有了，那就根据手感判断下盘子里应该有多少菜，然后把菜匀匀）；
4. 因为第二步和第三步的结果可能不是最优的，所以重复第二步和第三步，直到收敛（重复多次匀匀的过程，直到两个碟子中菜的量大致一样）。

而上面的第二步被称作 E 步（求期望），第三步被称作 M 步（求极大化），从而不断的 E、M。



EM 迭代优化的路径是直线式的，可以看到每一步都会向最优点前进一步，而且前进路线是平行于坐标轴的，因为每一步只优化一个变量。

这犹如在 x-y 坐标系中找一个曲线的极值，然而曲线函数不能直接求导，因此什么梯度下降方法就不适用了。但固定一个变量后，另外一个可以通过求导得到，因此可以使用坐标上升法，一次固定一个变量，对另外的求极值，最后逐步逼近极值。对应到 EM 上，E 步：固定  $\theta_A$ ，优化  $Q$ ；M 步：固定  $Q$ ，优化  $\theta_A$ ；交替将极值推向最大。

### 3.1.3 EM 和 MLE

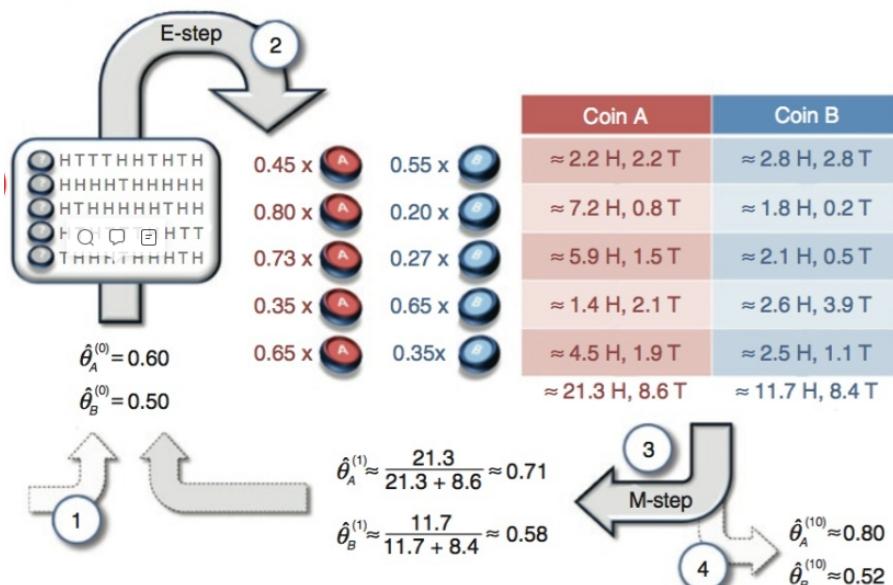
a Maximum likelihood

	Coin A	Coin B
		5 H, 5 T
	9 H, 1 T	
	8 H, 2 T	
		4 H, 6 T
	7 H, 3 T	
	24 H, 6 T	9 H, 11 T

$$\hat{\theta}_A = \frac{24}{24 + 6} = 0.80$$

$$\hat{\theta}_B = \frac{9}{9 + 11} = 0.45$$

b Expectation maximization



- MLE 是在“模型已定，参数未知”的情况下根据给定观察序列（所有序列服从同一分布）估计模型参数的估计方法。模型参数的准确性，跟观察序列直接相关。
- EM 算法就是含有隐变量的 MLE。

## 3.2 数学模型

假设现在有  $m$  个独立样本  $x = (x_1, x_2, \dots, x_m)$ , 模型参数为  $\theta$

$$\theta_{MLE} = \operatorname{argmax}_{\theta} \log p(x|\theta)$$

对于 EM 算法：

$$\theta^{(t+1)} = \operatorname{argmax}_{\theta} E_{z|x,\theta^{(t)}} [\log P(x, z|\theta)]$$

EM 算法的迭代过程为,  $\theta_0, \theta^{(1)}, \theta^{(1)}, \dots, \theta^{(n)}$  至收敛

下证：

$$P(x|\theta^{(t)}) \leq P(x|\theta^{(t+1)})$$

证明：

$$P(x|\theta) = \frac{P(x, z|\theta)}{P(z|x, \theta)}$$

$$\Rightarrow \log P(x|\theta) = \log P(x, z|\theta) - \log P(z|x, \theta) \quad (3.1)$$

$$\int_Z P(z|x, \theta^{(t)}) \log P(x|\theta) dz = \log P(x|\theta) \int_Z P(z|x, \theta^{(t)}) dz = \log P(x|\theta) \quad (3.2)$$

$$\int_Z P(z|x, \theta^{(t)}) [\log P(x, z|\theta) - \log P(z|x, \theta)] dz \quad (3.3)$$

$$= \int_Z P(z|x, \theta^{(t)}) \log P(x, z|\theta) dz - \int_Z P(z|x, \theta^{(t)}) \log P(z|x, \theta) dz \quad (3.4)$$

$$\text{令 } Q(\theta, \theta^{(t)}) = \int_Z P(z|x, \theta^{(t)}) \log P(x, z|\theta) dz$$

$$\text{令 } H(\theta, \theta^{(t)}) = \int_Z P(z|x, \theta^{(t)}) \log P(z|x, \theta) dz$$

则

$$\begin{aligned} \log P(x|\theta) &= Q(\theta, \theta^{(t)}) - H(\theta, \theta^{(t)}) \\ \log P(x|\theta^{(t)}) &= Q(\theta^{(t)}, \theta^{(t)}) - H(\theta^{(t)}, \theta^{(t)}) \end{aligned} \quad (3.5)$$

$$\log P(x|\theta^{(t+1)}) = Q(\theta^{(t+1)}, \theta^{(t)}) - H(\theta^{(t+1)}, \theta^{(t)}) \quad (3.6)$$

要证：

$$P(x|\theta^{(t)}) \leq P(x|\theta^{(t+1)})$$

等价于证：

$$\log P(x|\theta^{(t)}) \leq \log P(x|\theta^{(t+1)})$$

等价于证：

$$\begin{aligned} Q(\theta^{(t)}, \theta^{(t)}) &\leq Q(\theta^{(t+1)}, \theta^{(t)}) \\ H(\theta^{(t)}, \theta^{(t)}) &\geq H(\theta^{(t+1)}, \theta^{(t)}) \end{aligned} \quad (3.7)$$

$$\begin{aligned} \theta^{(t+1)} &= \operatorname{argmax}_{\theta} Q(\theta, \theta^{(t)}) \\ Q(\theta^{(t+1)}, \theta^{(t)}) &\geq Q(\theta, \theta^{(t)}) \end{aligned} \quad (3.8)$$

由  $\theta$  的任意性，知  $Q(\theta^{(t+1)}, \theta^{(t)}) \geq Q(\theta^{(t)}, \theta^{(t)})$

由 Jensen 不等式：  $E[\log X] \leq \log E[X]$

$$\begin{aligned} H(\theta^{(t+1)}, \theta^{(t)}) - H(\theta^{(t)}, \theta^{(t)}) \\ = \int_Z P(z|x, \theta^{(t)}) \log \frac{P(z|x, \theta^{(t+1)})}{P(z|x, \theta^{(t)})} dz \end{aligned} \quad (3.9)$$

$$= E[\log \frac{P(z|x, \theta^{(t+1)})}{P(z|x, \theta^{(t)})}] \quad (3.10)$$

$$\leq \log E(\frac{P(z|x, \theta^{(t+1)})}{P(z|x, \theta^{(t)})}) \quad (3.11)$$

$$= \log \int_Z P(z|x, \theta^{(t)}) \frac{P(z|x, \theta^{(t+1)})}{P(z|x, \theta^{(t)})} dz \quad (3.12)$$

$$= \log \int_Z P(z|x, \theta^{(t+1)}) dz \quad (3.13)$$

$$= \log 1 = 0 \quad (3.14)$$

### 3.3 EM 的应用

#### 3.3.1 EM 的优缺点

1. 优点：算法简单，稳定上升的步骤能非常可靠地找到“最优的收敛值”；
2. 缺点
  - (a) EM 算法的收敛速度，非常依赖初始值的设置，设置不当，计算时的代价是相当大的
  - (b) 对于大规模数据和多维高斯分布，其总的迭代过程，计算量大，迭代速度易受影响
  - (c) EM 算法中的 M-Step 依然是采用求导函数的方法，所以它找到的是极值点，即局部最优解，而不一定是全局最优解。

#### 3.3.2 EM 的应用

- K-means 聚类
- GMM(Gaussian Mixture Model, 高斯混合模型)

- HMM(Hidden Markov Model, 隐马尔可夫模型)

# Chapter 4

## 套索算法

### 4.1 背景

#### 4.1.1 线性模型的最小二乘估计

考虑线性模型：

$$y = X\beta + \epsilon, E(\epsilon) = 0, Cov(\epsilon) = \sigma^2 I \quad (4.1)$$

的参数  $\beta$  和  $\sigma^2$  的估计问题， $y$  为  $n*1$  观测向量， $X$  为  $n*p$  的设计矩阵， $\beta$  是  $p*1$  未知参数向量， $\epsilon$  为随机误差， $\sigma^2$  为误差的方差。

传统回归分析估计向量  $\beta$  的基本方法为最小二乘法，其思想是使得误差向量  $\epsilon = y - X\beta$  尽可能的小，使

$$Q(\beta) = ||\epsilon||^2 = ||y - X\beta||^2 = (y - X\beta)'(y - X\beta) \quad (4.2)$$

达到最小，解得

$$\hat{\beta} = (X'X)^{-1}X'y \quad (4.3)$$

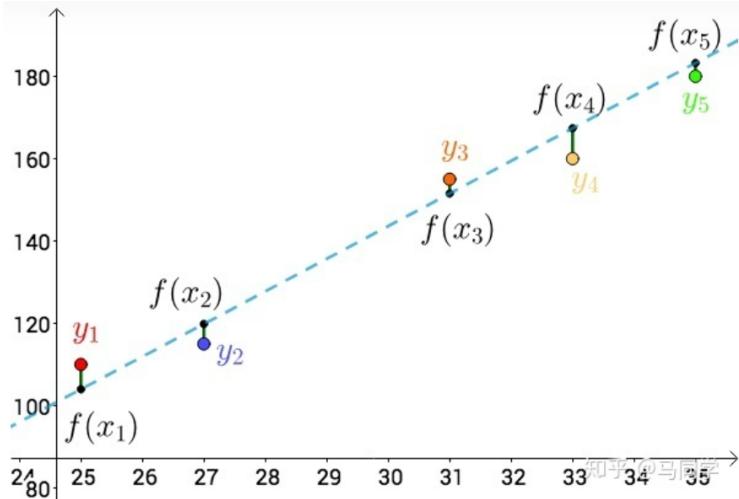
在传统的数据分析场景里，我们通常接触的数据大部分都是  $n>p$ ，即样本数大与变量数。此时，若  $\text{rank}(X)=p$ ，则  $X'X$  可逆，这时  $\hat{\beta}$  是  $\beta$  的无偏估计。

考察参数估计量的统计性质是衡量估计量好坏的主要准则。一个用于考察总体的估计量，可以从六个方面考察其优劣性。

估计量的统计性质	具体含义
线性	其是否是另一个随机变量的线性函数。
无偏性	其均值或期望是否等于总体的真实值。
有效性	其发生在所有的线性无偏估计量中具有最小方差。
渐近无偏性	样本容量趋于无穷大时，其均值序列趋于总体的真值。
一致性	样本容量趋于无穷大时，其是否依概率收敛于总体的真值。
渐近有效性	样本容量趋于无穷大时，其在所有的一致估计量中具有最小的渐近方差。

最小二乘估计有许多优良性质：线性、无偏性、最小方差性。

著名的高斯-马尔可夫 (Gauss-Markov) 定理：在古典假设条件下，最小二乘估计是最佳线性无偏估计量。



### 4.1.2 岭估计的提出

随着大数据的兴起，数据采集能力的指数级提升，大量的数据集出现了变量数多余样本数的情况，即  $p > n$ 。此时矩阵  $X$  出现多重共线性的情况， $X'X$  不可逆，因而没法用传统的 OLS（最小二乘估计）方法。

为了解决这一问题，就有了岭回归（Ridge Regression）方法，简单来说，岭回归就是在前面最小化目标函数  $Q(\beta)$  的后面加了一个 2-范数的平方：

$$Q(\beta) = \|y - X\beta\|^2 + \lambda\|\beta\|_2^2 \quad (4.4)$$

$$\Leftrightarrow \operatorname{argmin} \|y - X\beta\|^2 \quad s.t. \sum \beta_j^2 \leq s \quad (4.5)$$

上式求解可得到  $\beta$  的岭估计：

$$\hat{\beta}(\lambda) = (X'X + \lambda I)^{-1}X'y \quad (4.6)$$

从上面我们可以明显看到参数  $\lambda$  保证了  $X'X + \lambda I$  满秩、可逆，当然也由于其加入，此时的  $\hat{\beta}(\lambda)$  是一个有偏估计。

岭估计有以下性质：

1. 岭估计  $\hat{\beta}(\lambda)$  是  $\beta$  的有偏估计.
2. 岭估计  $\hat{\beta}(\lambda)$  是最小二乘估计  $\hat{\beta}$  的一个线性变换.
3.  $\forall k > 0$ , 若  $\|\hat{\beta}\| \neq 0 \Rightarrow \|\hat{\beta}(\lambda)\| < \|\hat{\beta}\|$ , 岭估计是一个压缩的有偏估计。
4. 存在  $\lambda > 0$ , 使得在均方误差意义下, 岭估计优于最小二乘估计, 即  $MSE(\hat{\beta}(\lambda)) < MSE(\hat{\beta})$

关于  $\lambda$  的选择，有 Horel-Kennard 公式、岭迹法、交叉验证法……

Horel-Kennard 公式：

## 4.2 LASSO

LASSO 全名 least absolute shrinkage and selection operator, 最小收缩算子法。他最大的特点就是引入了惩罚项，这个参数可以对模型变量进一步筛选，使模型不至于过于复杂，从而提高其泛化能力。

与岭回归类似，Lasso 就是在目标函数  $Q(\beta)$  后面加了一个 1-范数

$$Q(\beta) = \|y - X\beta\|^2 + \lambda\|\beta\|_1 \quad (4.7)$$

$$\Leftrightarrow \operatorname{argmin} \|y - X\beta\|^2 \quad s.t. \sum |\beta_j| \leq s \quad (4.8)$$

Hoerl 和 Kennard 提出的选择岭参数  $k$  的公式为

$$\hat{k} = \frac{\hat{\sigma}^2}{\max_i \hat{\alpha}_i^2}.$$

注意到，理论上的最优岭参数是下列方程的解：令  $f'(k) = 0$ ，则有

$$\begin{aligned} f'(k) &= -2\sigma^2 \sum_{i=1}^p \frac{\lambda_i}{(\lambda_i + k)^3} + 2k \sum_{i=1}^p \frac{\lambda_i \alpha_i^2}{(\lambda_i + k)^3} = 2 \\ &\sum_{i=1}^p \frac{\lambda_i (k\alpha_i^2 - \sigma^2)}{(\lambda_i + k)^3} = 0. \end{aligned}$$

若  $k\alpha_i^2 - \sigma^2 < 0$  对  $i = 1, 2, \dots, p$  都成立，则  $f'(k) < 0$ ，于是取

$$k^* = \frac{\hat{\sigma}^2}{\max_i \hat{\alpha}_i^2},$$

当  $0 < k < k^*$  时， $f'(k) < 0$  恒成立，因而  $f(k)$  在  $(0, k^*)$  上是单调递减函数。再由  $f(k)$  是  $(0, k^*)$  上的连续函数得到  $f(k^*) < f(0)$ 。用  $\hat{\alpha}_i$  和  $\hat{\sigma}^2$  代替  $\alpha_i$  和  $\sigma^2$  即可得到我们需要的岭参数  $\hat{k}$ 。

三种估计量的比较：

$$\begin{aligned} \text{Linear Regression : } &\sum_{i=1}^N \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 \\ \text{Lasso Regression : } &\sum_{i=1}^N \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \\ \text{Ridge Regression : } &\sum_{i=1}^N \left( y_i - \sum_{j=1}^p x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \end{aligned}$$

关于 LSAAO 在岭估计之后提出，为什么加 1-范数的 LASSO 没有加 2-范数的岭回归早，可能的原因是 1-范数作为绝对值之和不方便求导。

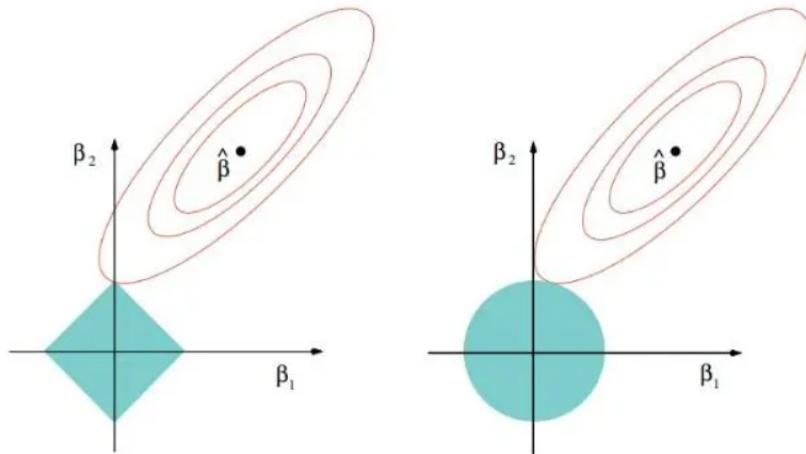
### 4.2.1 Lasso 的优势

为什么至今 Lasso 仍长青不老？

因为它可以解决现在高维数据一个普遍问题——稀疏性。高维数据即  $p > n$  的情况，现在随着数据采集能力的提高，变量（也叫特征）数采集的多，但是其中可能有很多特征是不重要的，系数很小，如果用岭回归，可能这个不重要的变量也给

你估出来了，而且可能还不小，而用 Lasso 方法，就可以把这些不重要变量的系数压缩为 0，既实现了较为准确的参数估计，也实现了变量选择（降维）。

以 Lasso 始祖 Tibshiran 在其著作中举的  $p=2$  的情形为例：



**FIGURE 3.11.** Estimation picture for the lasso (left) and ridge regression (right). Shown are contours of the error and constraint functions. The solid blue areas are the constraint regions  $|\beta_1| + |\beta_2| \leq t$  and  $\beta_1^2 + \beta_2^2 \leq t^2$ , respectively, while the red ellipses are the contours of the least squares error function.

图中青色部分表示两种方法的约束。

从图中可以看出，Lasso 方法与之相交的地方恰为  $(0, \beta_2)$ ，而从图中也可以看出  $\hat{\beta}$  所处的位置本就是  $\beta_2$  大， $\beta_1$  小，我们取 Lasso 的结果，意味着  $\beta_1$  的系数被压缩到了 0。

### 4.2.2 Lasso 最优解

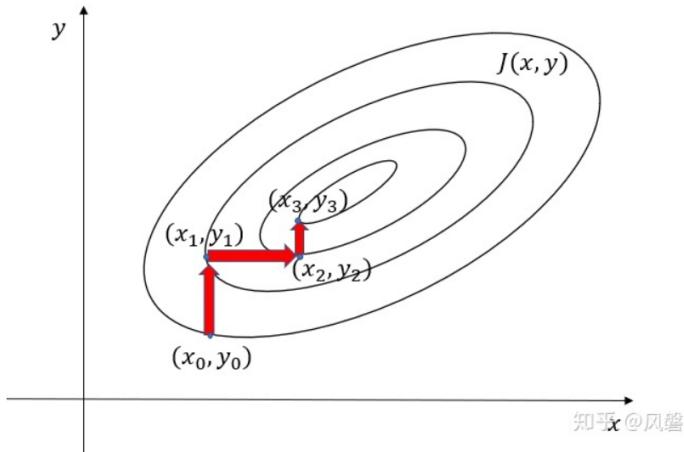
Lasso 因为其约束条件（也有叫损失函数的）不是连续可导的，因此常规的解法如梯度下降法、牛顿法、就没法用了。目前常用的方法有：坐标轴下降法与最小角回归法。

坐标轴下降法：

坐标轴下降法是一种迭代算法，与梯度下降法利用目标函数的导数来确定搜索方向不同，坐标轴下降法是在当前坐标轴上搜索函数最小值，不需要求目标函数

的导数。

以 2 维为例，设损失函数为凸函数  $J(x,y)$ ，在初始点固定  $x_0$ ，找使得  $J(y)$  达到最小的  $y_1$ ，然后固定  $y_1$ ，找使得  $J(x)$  达到最小的  $x_2$ ，这样一直迭代下去，因为  $J(x,y)$  是凸的，所以一定可以找到使得  $J(x,y)$  达到最小的点  $(x_k, y_k)$ 。



### 4.2.3 ElasticNet 模型

ElasticNet 模型，弹性网络模型，是结合了 lasso 和 ridge regression 的模型。

ElasticNet 模型的目标函数  $Q(\beta)$

$$Q(\beta) = \|y - X\beta\|^2 + \lambda[(1 - \alpha)\|\beta\|_2^2/2 + \alpha\|\beta\|_1] \quad (4.9)$$

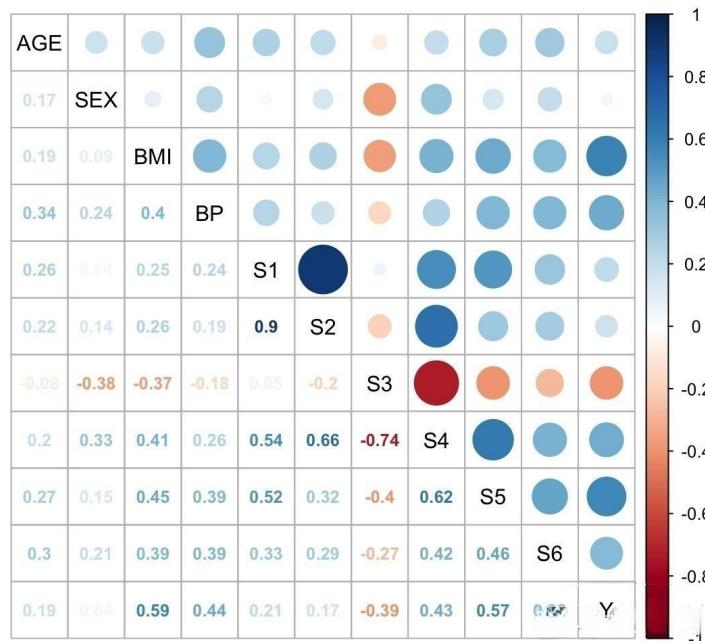
弹性网络惩罚由  $\alpha$  控制，当  $\alpha = 1$  时为 Lasso 模型，当  $\alpha = 0$  时为 Ridge 模型。

弹性网络在很多特征互相联系的情况下是非常有用的。Lasso 很可能只随机考虑这些特征中的一个，而弹性网络更倾向于选择两个。在实践中，Lasso 和 Ridge 之间权衡的一个优势是它允许在循环过程 (Under rotate) 中继承 Ridge 的稳定性。

## 4.3 算法实例

### 4.3.1 使用 R 进行 Ridge 回归

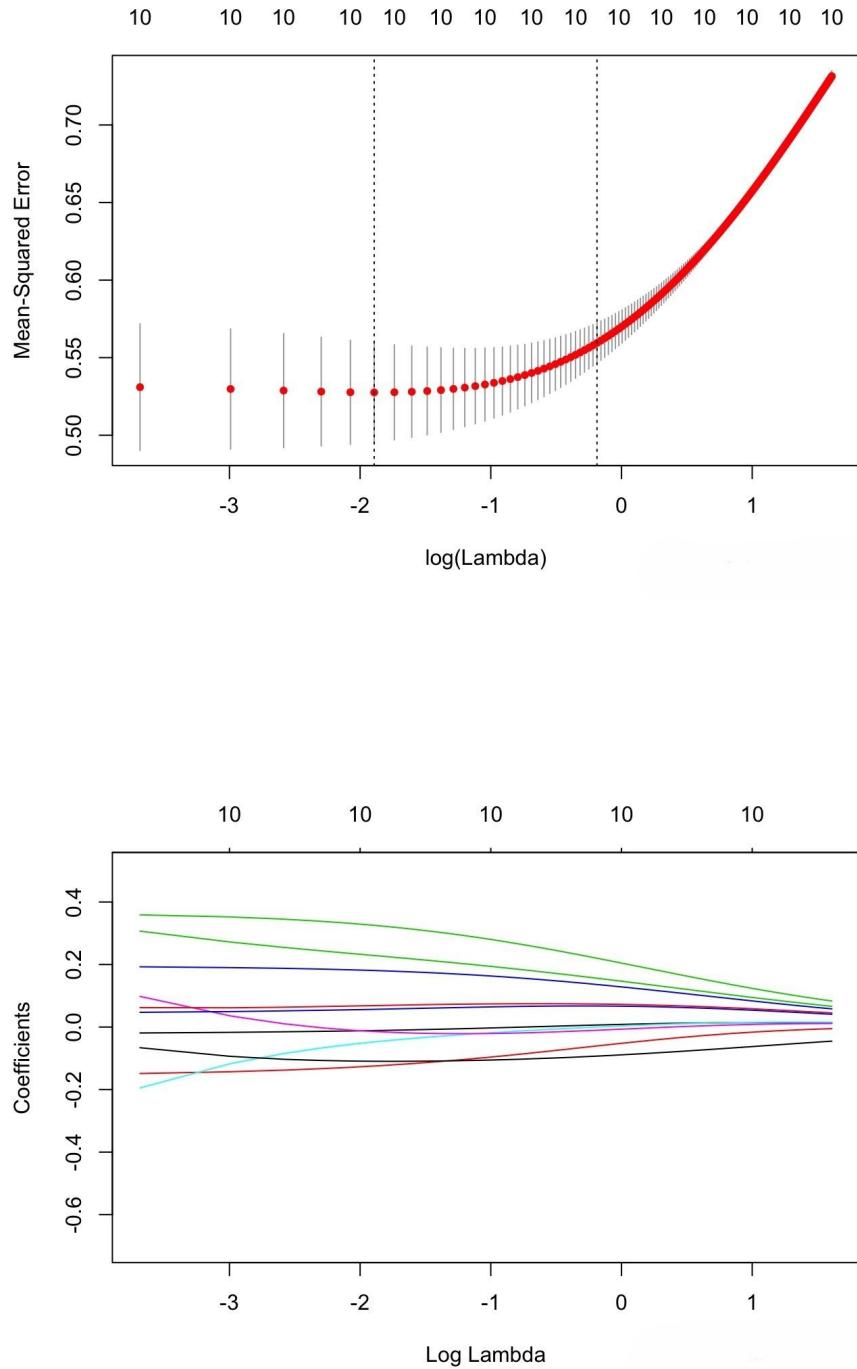
使用的数据集为糖尿病病情数据集（diabetes.csv）。糖尿病病情数据集包含从 442 例糖尿病患者中获得的十个变量：年龄（AGE）、性别（SEX）、体重指数（BMI）、平均血压（BP）和六个血清测量值（S1-S6），以及一个我们感兴趣的因变量 Y。得到的可视化相关系数如图 1 所示：



数据相关性分析结果表明，因变量 Y 和 AGE、SEX、S1、S2 四个变量的相关系数较小，和 S3 是负相关，与其余变量的相关系数较大，且都是正相关；S1 和 S2 之间的正相关性较强，S3 和 S4 的负相关性较强。

在 Ridge 回归模型中，需要指定一个合适的参数 lambda，该参数为施加在回归系数上的惩罚系数，不同的参数 lambda 可以得到不同的 Ridge 回归模型。glmnet 包中的 cv.glmnet() 可以通过交叉验证的方式，来分析在不同的参数 lambda 下回归模型的效果。下面使用 cv.glmnet() 函数和训练数集分析模型参数的影响。

通过建立 Ridge 回归，在不同的 lambda 下，变量合数、各个自变量回归系数的变化如下图所示：

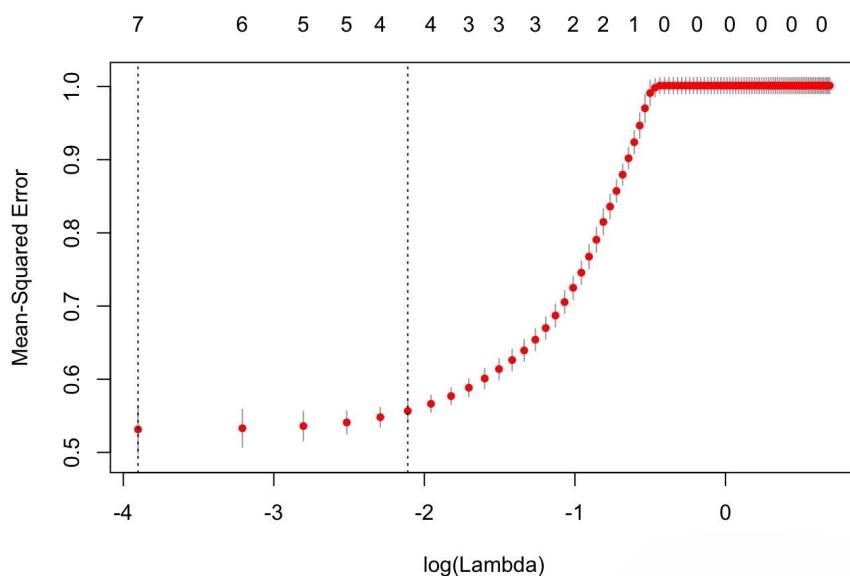


结果表明，不存在系数为 0 的自变量，这是因为 Ridge 模型不容易把变量的系数压缩到 0，Ridge 回归模型并没有自动进行变量选择的能力。为了验证 Ridge 回归模型的预测效果，可在测试集上进行预测，并计算出平均绝对值误差的大小。通过 predict() 函数得到测试集的预测值后，再使用 Metrics 包的 mae() 函数计算平均绝对值误差。通过将预测值进行逆标准化变换，与标准化前的因变量进行比较发现，Ridge 回归模型的预测平均绝对值误差为 45.99。

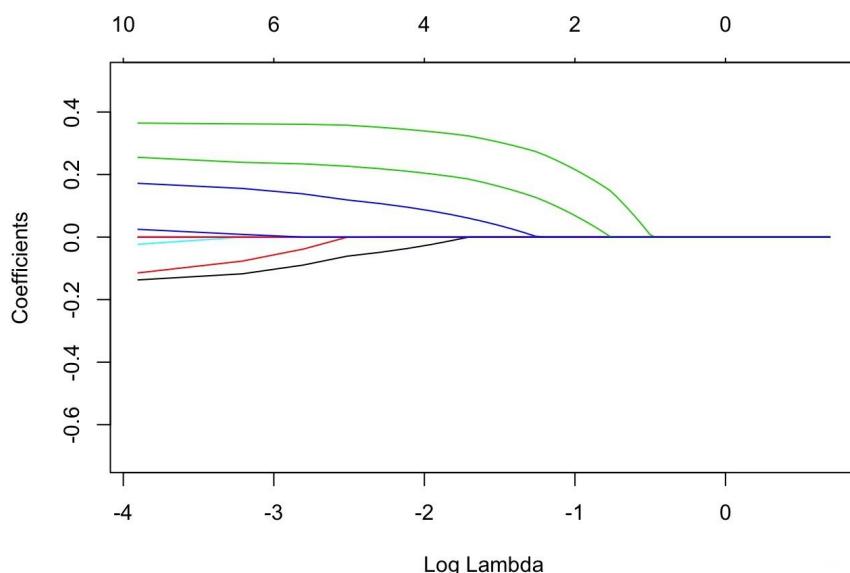
### 4.3.2 使用 R 进行 Lasso 回归

使用同样的数据集进行 Lasso 回归分析。

使用 cv.glmnet() 函数，利用交叉验证的方式，分析不同的参数 lambda 下 Lasso 回归模型的效果。在得到交叉验证后的模型 lasso model 后，使用 plot() 函数可视化不同参数下的均方误差以及各个自变量的回归系数变化情况，结果下图所示。



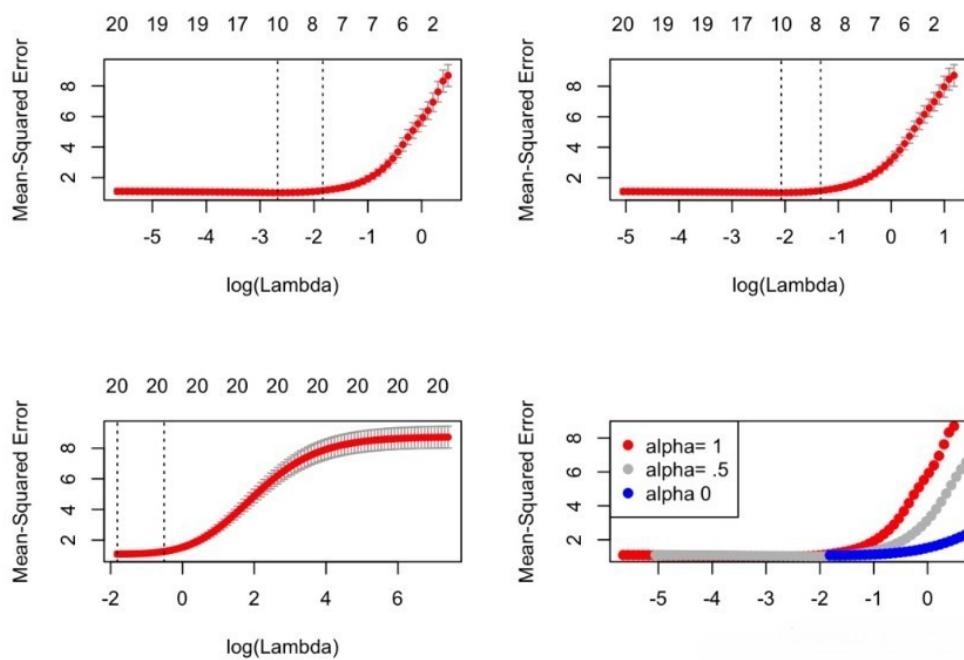
随着参数 lambda 值的增加，Lasso 回归使用的自变量数目在减少，同时模型的预测误差在增大。从 Lasso 回归模型的系数可以发现，AGE、S2、S4 三个自变量的回归系数等于 0，说明模型将这 3 个对因变量影响不显著的特征剔除了。



通过 `predict()` 函数得到测试集的预测值后，再使用 Metrics 包的 `mae()` 函数计算平均绝对值误差。通过将预测值进行逆标准化变换，与标准化前的因变量进行比较发现，Lasso 回归模型的预测平均绝对值误差为 46.22。结果发现，Lasso 回归预测的误差比 Ridge 回归大了一点，这是因为 Lasso 回归使用了更少的自变量来建立回归模型。虽然误差稍微变大，但是 Lasso 使用更少的特征，使模型更稳定，降低了模型的复杂度。

### 4.3.3 ElasticNet 模型

ElasticNet 模型，弹性网络模型，是结合了 lasso 和 ridge regression 的模型。弹性网络惩罚由  $\alpha$  控制，当  $\alpha = 1$  时为 Lasso 模型，当  $\alpha = 0$  时为 Ridge 模型。



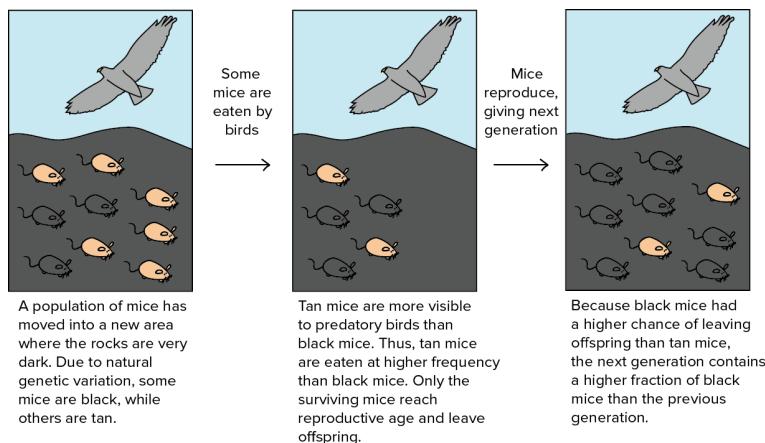
# Chapter 5

## 自然进化策略算法

### 5.1 Evolution Strategy

Evolution strategies (ES) belong to the big family of evolutionary algorithms. The optimization targets of ES are vectors of real numbers,  $x \in R^n$ .

Evolutionary algorithms refer to a division of population-based optimization algorithms inspired by natural selection. Natural selection believes that individuals with traits beneficial to their survival can live through generations and pass down the good characteristics to the next generation. Evolution happens by the selection process gradually and the population grows better adapted to the environment.



Evolutionary algorithms can be summarized in the following format as a general optimization solution:

Let's say we want to optimize a function  $f(x)$  and we are not able to compute gradients directly. But we still can evaluate  $f(x)$  given any  $x$  and the result is deterministic. Our belief in the probability distribution over  $x$  as a good solution to  $f(x)$  optimization is  $p_\theta(x)$ , parameterized by  $\theta$ . The goal is to find an optimal configuration of  $\theta$ .

Starting with an initial value of  $\theta$ , we can continuously update  $\theta$  by looping three steps as follows:

1. Generate a population of samples  $D = (x_i, f(x_i))$  where  $x_i \sim p_\theta(x)$
2. Evaluate the "fitness" of samples in  $D$ .
3. Select the best subset of individuals and use them to update  $\theta$ , generally bases on fitness or rank.

The most basic and canonical version of evolution strategies: Simple Gaussian Evolution Strategies. It models  $p_\theta(x)$  as a n-dimensional isotropic Gaussian distribution, in which  $\theta$  only tracks the mean  $\mu$  and standard deviation  $\sigma$ .

$$\theta = (\mu, \sigma), p_\theta(x) \sim N(\mu, \sigma^2 I) = \mu + \sigma N(0, I) \quad (5.1)$$

The process of Simple-Gaussian-ES, given  $x \in R^n$ :

1. Initialize  $\theta = \theta^{(0)}$  and the generation counter  $t=0$
2. Generate the offspring population of size  $\Lambda$  by sampling from the Gaussian distribution:

$$D^{(t+1)} = x_i^{(t+1)} | x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)} \text{ where } y_i^{(t+1)} \sim N(0, I); i = 1, \dots, \Lambda \quad (5.2)$$

3. Select a top subset of  $\lambda$  samples with optimal  $f(x_i)$  and this subset is called **elite** set. Let's label them as

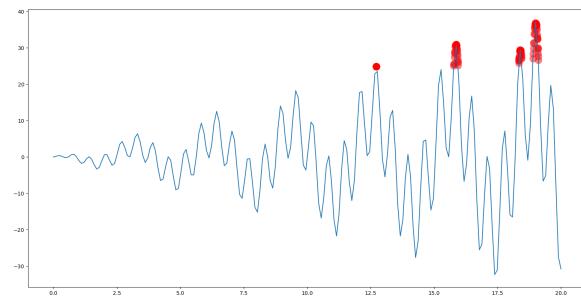
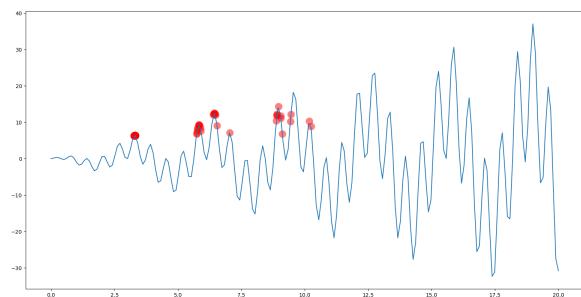
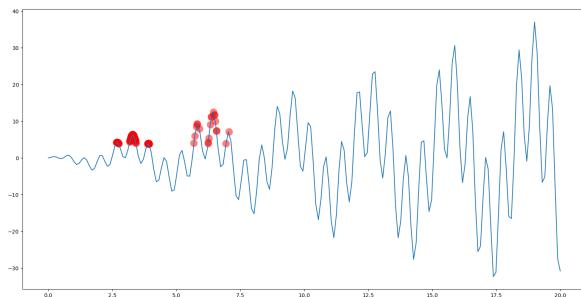
$$D_{elite}^{(t+1)} = x_i^{(t+1)} | x_i^{(t+1)} \in D^{(t+1)}, i = 1, \dots, \lambda, \lambda \leq \Lambda \quad (5.3)$$

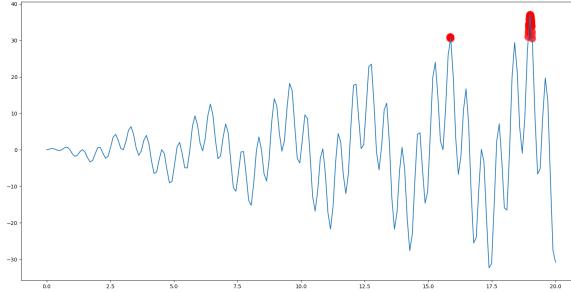
4. Then we estimate the new mean and std for the next generation using the elite set:

$$\mu^{(t+1)} = \text{avg}(D_{elite}^{(t+1)}) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} x_i^{(t+1)} \quad (5.4)$$

$$\sigma^{(t+1)^2} = \text{var}(D_{elite}^{(t+1)}) = \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})^2 \quad (5.5)$$

5. Repeat steps 2-4 until the result is good enough.





## 5.2 CMA Evolution Strategy

The standard deviation  $\sigma$  accounts for the level of exploration: the larger  $\sigma$  the bigger search space we can sample our offspring population. In vanilla ES,  $\sigma^{(t+1)}$  is highly correlated with  $\sigma^t$ , so the algorithm is not able to rapidly adjust the exploration space when needed (i.e. when the confidence level changes).

CMA-ES, short for “Covariance Matrix Adaptation Evolution Strategy”, fixes the problem by tracking pairwise dependencies between the samples in the distribution with a covariance matrix  $C$ . The new distribution parameter becomes:

$$\theta = (\mu, \sigma, C), p_\theta(x) \sim N(\mu, \sigma^2 C) \sim \mu + \sigma N(0, C) \quad (5.6)$$

where  $\sigma$  controls for the overall scale of the distribution, often known as step size.

Compared with ES mentioned above:

$$\theta = (\mu, \sigma), p_\theta(x) \sim N(\mu, \sigma^2 I) = \mu + \sigma N(0, I) \quad (5.7)$$

### 5.2.1 Updating the Mean

$$\mu^{(t+1)} = \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)}) \quad (5.8)$$

CMA-ES has a learning rate  $\alpha_\mu \leq 1$  to control how fast the mean should be updated.

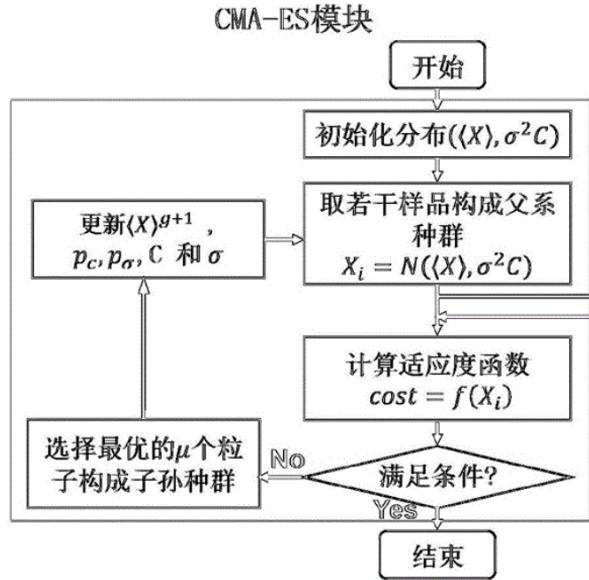
Symbol	Meaning
$x_i^{(t)} \in \mathbb{R}^n$	the $i$ -th samples at the generation (t)
$y_i^{(t)} \in \mathbb{R}^n$	$x_i^{(t)} = \mu^{(t-1)} + \sigma^{(t-1)} y_i^{(t)}$
$\mu^{(t)}$	mean of the generation (t)
$\sigma^{(t)}$	step size
$C^{(t)}$	covariance matrix
$B^{(t)}$	a matrix of $C$ 's eigenvectors as row vectors
$D^{(t)}$	a diagonal matrix with $C$ 's eigenvalues on the diagnose.
$p_\sigma^{(t)}$	evaluation path for $\sigma$ at the generation (t)
$p_c^{(t)}$	evaluation path for $C$ at the generation (t)
$\alpha_\mu$	learning rate for $\mu$ 's update
$\alpha_\sigma$	learning rate for $p_\sigma$
$d_\sigma$	damping factor for $\sigma$ 's update
$\alpha_{cp}$	learning rate for $p_c$
$\alpha_{c\lambda}$	learning rate for $C$ 's rank-min( $\lambda$ , n) update
$\alpha_{c1}$	learning rate for $C$ 's rank-1 update

**Algorithm 1** CMA-ES: Covariance Matrix Adaptation Evolution Strategies

---

**Require:**  $\alpha_\mu, \alpha_\sigma, \alpha_{cp}, \alpha_{c1}, \alpha_{c\lambda}$  ▷ Learning rates  
**Require:**  $d_\sigma$  ▷ Damping factor  
**Require:**  $t = 0$  ▷ Generation counter  
**Require:**  $\mu^{(0)} \in \mathbb{R}^n, \sigma \in \mathbb{E}_+$  ▷ Inputs: initial mean vectors and step size  
**Require:**  $C^{(0)} = I, p_\sigma^{(0)} = 0, p_c^{(0)} = 0$  ▷ Initialize covariance matrix and evolution paths.  
**repeat**  
    Sample  $x_i^{(t+1)} = \mu^{(t)} + {}^{(t)} y_i$  where  $y_i \sim \mathcal{N}(0, C^{(t)})$ ,  $i = 1, \dots, \Lambda$   
    Select top  $\lambda$  samples with the best performance  $x_i^{(t+1)}, i = 1, \dots, \lambda$   
     $\mu^{(t+1)} \leftarrow \mu^{(t)} + \alpha_\mu \frac{1}{\lambda} \sum_{i=1}^{\lambda} (x_i^{(t+1)} - \mu^{(t)})$   
     $p_\sigma^{(t+1)} \leftarrow (1 - \alpha_\sigma)p_\sigma^{(t)} + \sqrt{\alpha_\sigma(2 - \alpha_\sigma)\lambda} C^{(t)} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$   
     $\sigma^{(t+1)} \leftarrow \sigma^{(t)} \exp \left( \frac{\alpha_\sigma}{d_\sigma} \left( \frac{\|p_\sigma^{(t+1)}\|}{\mathbb{E}\|\mathcal{N}(0, I)\|} - 1 \right) \right)$   
     $p_c^{(t+1)} \leftarrow (1 - \alpha_{cp})p_c^{(t)} + \sqrt{\alpha_{cp}(2 - \alpha_{cp})\lambda} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}}$   
     $C^{(t+1)} \leftarrow (1 - \alpha_{c1})C^{(t)} + \alpha_{c1} p_c^{(t+1)} p_c^{(t+1)^\top}$   
     $C^{(t+1)} \leftarrow (1 - \alpha_{c\lambda} - \alpha_{c1})C^{(t)} + \alpha_{c1} p_c^{(t+1)} p_c^{(t+1)^\top} + \alpha_{c\lambda} \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)^\top}$   
     $t \leftarrow t + 1$   
**until** hit stopping criteria  
**return**  $\mu^{(t)}, \sigma^{(t)}, C^{(t)}$

---



### 5.2.2 Controlling the Step Size

The sampling process can be decoupled from the mean and standard deviation:

$$x_i^{(t+1)} = \mu^{(t)} + \sigma^{(t)} y_i^{(t+1)}, \text{ where } y_i^{(t+1)} = \frac{x_i^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \sim N(0, C) \quad (5.9)$$

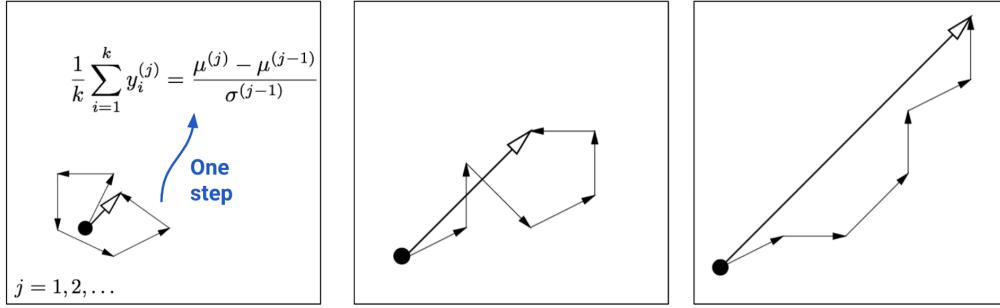
The parameter  $\sigma$  controls the overall scale of the distribution. It is separated from the covariance matrix so that we can change steps faster than the full covariance. A larger step size leads to faster parameter update. In order to evaluate whether the current step size is proper, CMA-ES constructs an evolution path  $p_\sigma$  by summing up a consecutive sequence of moving steps.

By comparing this path length with its expected length under random selection (meaning single steps are uncorrelated), we are able to adjust  $\sigma$  accordingly.

Single steps cancel each other off and thus evolution path is short.  
 $\rightarrow$  Decrease  $\sigma$

Ideal case: single steps are uncorrelated.

Single steps point to the same direction and thus evolution path is long.  
 $\rightarrow$  Increase  $\sigma$



$$\frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} = \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \quad (5.10)$$

$$\frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} \sim \frac{1}{\lambda} N(0, \lambda C^{(t)}) \sim \frac{1}{\sqrt{\lambda}} C^{(t)^{\frac{1}{2}}} N(0, I) \quad (5.11)$$

$$\implies \sqrt{\lambda} C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \sim N(0, I) \quad (5.12)$$

In order to assign higher weights to recent generations, we use polyak averaging to update the evolution path with learning rate  $\alpha_\sigma$ . Meanwhile, the weights are balanced so that  $p_\sigma$  is conjugate,  $\sim N(0, I)$  both before and after one update.

$$p_\sigma^{(t+1)} = (1 - \alpha_\sigma) p_\sigma^{(t)} + \sqrt{c_\sigma(2 - \alpha_\sigma) \lambda} C^{(t)^{-\frac{1}{2}}} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \quad (5.13)$$

The expected length of  $p_\sigma$  under random selection is  $E|N(0, I)|$ , that is the expectation of the L2-norm of a  $N(0, I)$  random variable.

We adjust the step size according to the ratio of  $\frac{\|p_\sigma^{(t+1)}\|}{E\|N(0, I)\|}$

$$\ln\sigma^{(t+1)} = \ln\sigma^{(t)} + \frac{\alpha_\sigma}{d_\sigma} \left( \frac{\|p_\sigma^{(t+1)}\|}{E\|N(0, I)\|} - 1 \right) \quad (5.14)$$

$$\sigma^{(t+1)} = \sigma^{(t)} \exp \left( \frac{\alpha_\sigma}{d_\sigma} \left( \frac{\|p_\sigma^{(t+1)}\|}{E\|N(0, I)\|} - 1 \right) \right) \quad (5.15)$$

where  $d_\sigma \approx 1$  is a damping parameter, scaling how fast  $\ln\sigma$  should be changed.

### 5.2.3 Adapting the Covariance Matrix

For the covariance matrix, it can be estimated from scratch using  $y_i$  of elite samples(recall that  $y_i \sim N(0, C)$ )

$$C_\lambda^{(t+1)} = \frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)T} \quad (5.16)$$

The above estimation is only reliable when the selected population is large enough. However, we do want to run fast iteration with a small population of samples in each generation. That's why CMA-ES invented a more reliable but also more complicated way to update C.

It involves two independent routes:

- **Rank-min( $\lambda, n$ ) update:**uses the history of  $C_\lambda$ , each estimated from scratch in one generation.
- **Rank-one update:**estimates the moving steps  $y_i$  and the sign information from the history.

1. For the first part:

The first route considers the estimation of C from the entire history of  $C_\lambda$ . For example, if we have experienced a large number of generations,  $C^{(t+1)} \approx \text{avg}(C_\lambda^{(i)}; i = 1, \dots, t)$  would be a good estimator. Similar to  $p_\sigma$ , we also use

polyak averaging with a learning rate to incorporate the history:

$$C^{(t+1)} = (1 - \alpha_{c\lambda})C^{(t)} + \alpha_{c\lambda}C^{(t+1)} \quad (5.17)$$

$$C^{(t+1)} = (1 - \alpha_{c\lambda})C^{(t)} + \alpha_{c\lambda}\frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)T} \quad (5.18)$$

A common choice for the learning rate is  $\alpha_{c\lambda} \approx \min(1, \lambda/n^2)$ .

2. For the second part:

Similar to how we adjust the step size  $\sigma$ , an evolution path  $p_c$  is used to track the sign information and it is constructed in a way that  $p_c$  is conjugate,  $\sim N(0, C)$  both before and after a new generation.

We may consider  $p_c$  as another way to compute  $avg_i(y_i)$  (notice that both  $\sim N(0, C)$ ) while the entire history is used and the sign information is maintained.

In the last section, we know that:

$$\sqrt{\lambda} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \sim N(0, C) \quad (5.19)$$

We define:

$$p_c^{(t+1)} = (1 - \alpha_{cp})p_c^{(t)} + \sqrt{1 - (1 - \alpha_{cp})^2} \sqrt{\lambda} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \quad (5.20)$$

$$p_c^{(t+1)} = (1 - \alpha_{cp})p_c^{(t)} + \sqrt{\alpha_{cp}(2 - \alpha_{cp})\lambda} \frac{\mu^{(t+1)} - \mu^{(t)}}{\sigma^{(t)}} \quad (5.21)$$

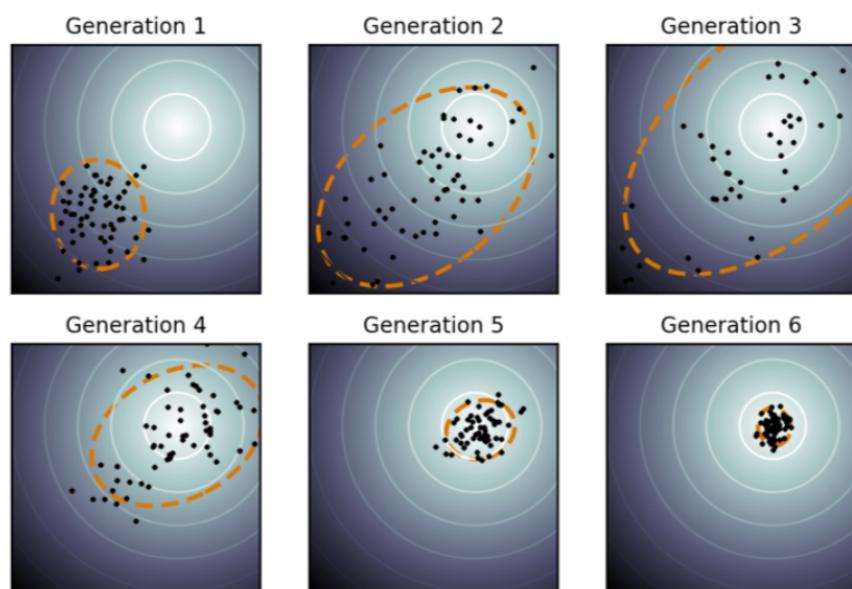
Then the covariance matrix is updated according to  $p_c$ :

$$C^{(t+1)} = (1 - \alpha_{c1})C^{(t)} + \alpha_{c1}p_c^{(t+1)} p_c^{(t+1)T} \quad (5.22)$$

Eventually we combine two approaches together:

Black dots are samples in one generation. The samples are more spread out initially but when the model has higher confidence in finding a good solution in the late stage, the samples become very concentrated over the global optimum.

$$C^{(t+1)} = (1 - \alpha_{c\lambda} - \alpha_{c1})C^{(t)} + \alpha_{c1} \underbrace{p_c^{(t+1)} p_c^{(t+1)\top}}_{\text{rank-one update}} + \alpha_{c\lambda} \underbrace{\frac{1}{\lambda} \sum_{i=1}^{\lambda} y_i^{(t+1)} y_i^{(t+1)\top}}_{\text{rank-min(lambda, n) update}}$$



# Chapter 6

## 深度强化学习

### 6.1 前言

今天介绍的主题为强化学习中的一种 actor critic 的提升方式 Deep Deterministic Policy Gradient (DDPG) 算法，DDPG 最大的优势就是能够在连续动作上更有效地学习。

概括来说，RL 要解决的问题是：让 agent 学习在一个环境中的如何行为动作 (act)，从而获得最大的奖励值总和 (total reward)。这个奖励值一般与 agent 定义的任务目标关联。agent 需要的主要学习内容：第一是行为策略 (action policy)，第二是规划 (planning)。其中，行为策略的学习目标是最优策略，也就是使用这样的策略，可以让 agent 在特定环境中的行为获得最大的奖励值，从而实现其任务目标。

行为 (action) 可以简单分为：

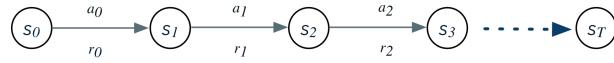
- 连续的：如赛车游戏中的方向盘角度、油门、刹车控制信号，机器人的关节伺服电机控制信号。
- 离散的：如围棋、贪吃蛇游戏。Alpha Go 就是一个典型的离散行为 agent。

DDPG 是针对连续行为的策略学习方法。在 RL 领域，DDPG 主要从：PG -> DPG -> DDPG 发展而来。

相关基本概念：

1.  $s_t$ : 在 t 时刻, agent 观察到的环境状态, 比如观察到的环境图像, agent 在环境中的位置、速度、机器人关节角度等;
2.  $a_t$ : 在 t 时刻, agent 选择的行为 (action), 通过环境执行后, 环境状态由  $s_t$  转换为  $s_{t+1}$
3.  $r(s_t, a_t)$  函数: 环境在状态  $s_t$  执行行为  $a_t$  后, 返回的单步奖励值;
4.  $R_t$ : 是从当前状态直到将来某个状态, 期间所有行为所获得奖励值的加权总和, 即 discounted future reward
5.  $\gamma$ : discounted rate, 通常取 0.99.

$$R_t = \sum_{i=t}^T \gamma^{i-t} r(s_i, a_i) \quad (6.1)$$



### 6.1.1 PG

R.Sutton 在 2000 年提出的 Policy Gradient 方法, 是 RL 中, 学习连续的行为控制策略的经典方法, 其提出的解决方案是：通过一个概率分布函数  $\pi_\theta(s_t|\theta^\pi)$  来表示每一步的最优策略, 在每一步根据该概率分布进行 action 采样, 获得当前的最佳 action 取值; 即：

$$a_t \sim \pi_\theta(s_t|\theta^\pi) \quad (6.2)$$

生成 action 的过程, 本质上是一个随机过程; 最后学习到的策略, 也是一个随机策略 (stochastic policy).

### 6.1.2 DPG

Deepmind 的 D.Silver 等在 2014 年提出 DPG: Deterministic Policy Gradient, 即确定性的行为策略, 每一步的行为通过函数  $\mu$  直接获得确定的值:

$$a_t = \mu(s_t | \theta^\mu) \quad (6.3)$$

这个函数  $\mu$  即最优行为策略, 不再是一个需要采样的随机策略。

为何需要确定性的策略? 简单来说, PG 方法有以下缺陷:

1. 即使通过 PG 学习得到了随机策略之后, 在每一步行为时, 我们还需要对得到的最优策略概率分布进行采样, 才能获得 action 的具体值; 而 action 通常都是高维的向量, 比如 25 维、50 维, 在高维的 action 空间的频繁采样, 无疑是很耗费计算能力的;
2. 在 PG 的学习过程中, 每一步计算 policy gradient 都需要在整个 action space 进行积分:

$$\nabla_\theta = \int_S \int_A \rho(s) \pi_\theta(a|s) Q^\pi(s, a) da ds \quad (6.4)$$

这个积分我们一般通过 Monte Carlo 采样来进行估算, 需要在高维的 action 空间进行采样, 耗费计算能力。

### 6.1.3 DDPG

Deepmind 在 2016 年提出 DDPG, 全称是: Deep Deterministic Policy Gradient, 是将深度学习神经网络融合进 DPG 的策略学习方法。相对于 DPG 的核心改进是: 采用卷积神经网络作为策略函数  $\mu$  和 Q 函数的模拟, 即策略网络和 Q 网络; 然后使用深度学习的方法来训练上述神经网络。

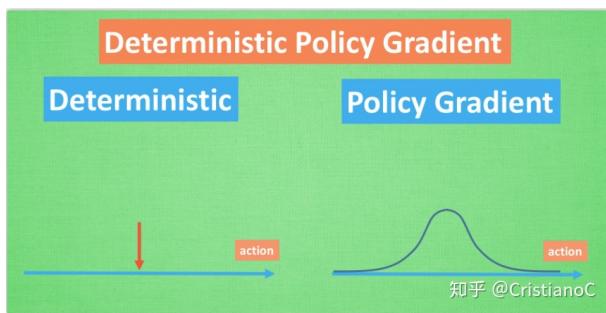
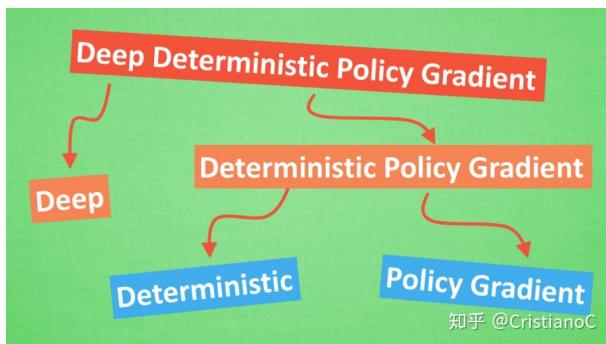
Q 函数的实现和训练方法, 采用了 Deepmind 2015 年发表的 DQN 方法, 即 Alpha Go 使用的 Q 函数方法。

## 6.2 DDPG 算法

### 6.2.1 算法背景

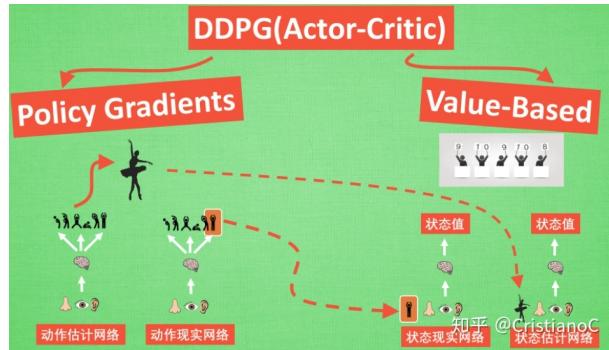
DDPG 算法它吸收了 Actor-Critic 让 Policy gradient 单步更新的精华，而且还吸收让计算机学会玩游戏的 DQN 的精华，合并成了一种新算法，叫做 Deep Deterministic Policy Gradient. 那 DDPG 到底是什么样的算法呢，我们就拆开来分析。我们将 DDPG 分成'Deep' 和'Deterministic Policy Gradient'，然后'Deterministic Policy Gradient' 又能细分为'Deterministic' 和'Policy Gradient'，接下来我们就开始一个个分析。

Deep 顾名思义，就是走向更深层次，我们在 DQN 中提到过，我们使用一个经验池和两套结构相同，但参数更新频率不同的神经网络能有效促进学习。那我们也把这种思想运用到 DDPG 中，使 DDPG 也具备这种优良形式。但是 DDPG 的神经网络形式却比 DQN 的要复杂一点。



Policy Gradient 相比其他的强化学习方法，它能被用来在连续动作上进行动作的筛选。而且筛选的时候是根据所学习到的动作分布随机进行筛选，而 Deterministic 有点看不下去，Deterministic 说：我说兄弟，你其实在做动作的时候没必要那

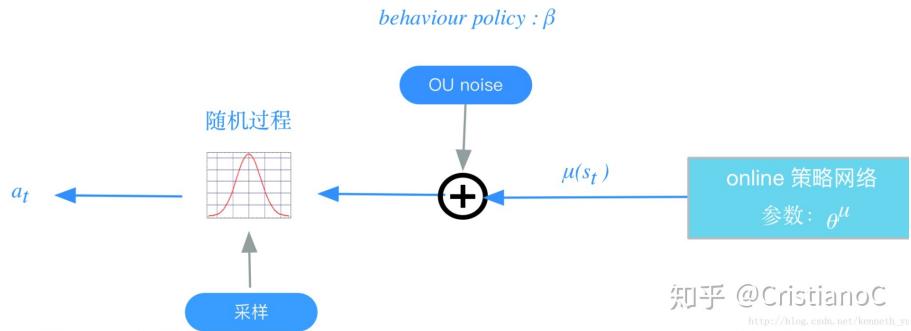
么不确定，反正你最终都只是要输出一个动作值，干嘛要随机。所以 Deterministic 就改变了输出动作的过程，只在连续动作上输出一个动作值。



现在我们来说说 DDPG 中所用到的神经网络（粗略）。它其实和我们之前提到的 Actor-Critic 形式差不多，也需要有基于策略 Policy 的神经网络和基于价值 Value 的神经网络。但是为了体现 DQN 的思想，每种神经网络都需要再细分成两个，Policy Gradient 这边，我们有估计网络和现实网络，估计网络用来输出实时的动作，供 actor 在现实中实行。而现实网络则是用来更新价值网络系统的。所以我们再来看看价值系统这边，我们也有现实网络和估计网络，他们都在输出这个状态的价值，而输入端却有不同，状态估计网络则是拿着当时 Actor 施加的动作当做输入。在实际运用中，DDPG 这种做法确实带来了更有效的学习过程。

### 6.2.2 算法相关概念和定义

- 确定性行为策略  $\mu$ : 定义为一个函数，每一步的行为可以通过  $s_t = \mu(s_t)$  计算获得。
- 策略网络：用一个卷积神经网络对  $\mu$  函数进行模拟拟合，这个网络我们就叫做策略网络，其参数为  $\theta^\mu$ ；
- behavior policy  $\beta$ : 在 RL 训练过程中，我们要兼顾两个 e: exploration 和 exploit (也就是之前说过的探索和开发)；exploration 的目的是探索潜在的更优策略，所以训练过程中，我们为 action 的决策机制引入随机噪声：将 action 的决策从确定性的过程变为一个随机过程，再从这个随机过程中采样得到 action，下达给环境执行，过程如下图所示：上述这个策略叫做 behavior



策略，用  $\beta$  来表示，这时 RL 的训练方式叫做 off-policy。这里与  $\epsilon$ -greedy 的思路是类似的。DDPG 中，使用 Uhlenbeck-Ornstein 随机过程（下面简称 UO 过程），作为引入的随机噪声：UO 过程在时序上具备很好的相关性，可以使 agent 很好的探索具备动量属性的环境。

4. Q 函数：即 action-value 函数，定义在状态  $s_t$  下，采取动作  $a_t$  后，且如果持续执行策略  $\mu$  的情况下，所获得的  $R_t$  期望值，用 Bellman 等式来定义：

$$Q^\mu(s_t, a_t) = E[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))] \quad (6.5)$$

可以看到，Q 函数的定义是一个递归表达，在实际情况中，我们不可能每一步都递归计算 Q 的值，可行的方案是通过一个函数对 Bellman 等式表达进行模拟。

5. Q 网络：DDPG 中，我们用一个卷积神经网络对 Q 进行模拟，这个网络我们就叫做 Q 网络，其参数为  $\theta^Q$ ，采用了 DQN 相同的方法。
6. 如何衡量一个策略  $\mu$  的表现：用一个函数  $J$  来衡量，我们叫做 performance objective，针对 off-policy 学习的场景。
7. 训练的目标：最大化  $J_\beta(\mu)$ ，同时最小化 Q 网络的 Loss,  $\mu = argmax_\mu J(\mu)$
8. 最优 Q 网络定义：具备最小化的 Q 网络 Loss；训练 Q 网络的过程，就是寻找 Q 网络参数的最优解的过程，我们使用 SGD 的方法。

### 6.2.3 DDPG 实现框架和算法

以往的实践证明，如果只使用单个 Q 神经网络的算法，学习过程很不稳定，因为 Q 网络的参数在频繁梯度更新的同时，又用于计算 Q 网络和策略网络的 gradient。基于此，DDPG 分别为策略网络、Q 网络各创建两个神经网络拷贝，一个叫做 online，一个叫做 target：

$$\text{策略网络} \begin{cases} \text{online : } \mu(s|\theta^\mu) : \text{gradient update } \theta^\mu \\ \text{target : } \mu'(s|\theta^{\mu'}) : \text{soft update } \theta^{\mu'} \end{cases}$$

$$Q\text{网络} \quad \begin{cases} \text{online : } Q(s, a|\theta^Q) : \text{gradient update } \theta^Q \\ \text{target : } Q'(s, a|\theta^{Q'}) : \text{soft update } \theta^{Q'} \end{cases}$$

在训练完一个 mini-batch 的数据之后，通过 SGA/SGD 算法更新 online 网络的参数，然后通过 soft update 算法更新 target 网络的参数。soft update 是一种 running average 的算法：

$$\text{soft update : } \begin{cases} \theta^{Q'} \leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} \leftarrow \tau \theta_\mu + (1 - \tau) \theta^{\mu'} \end{cases}$$

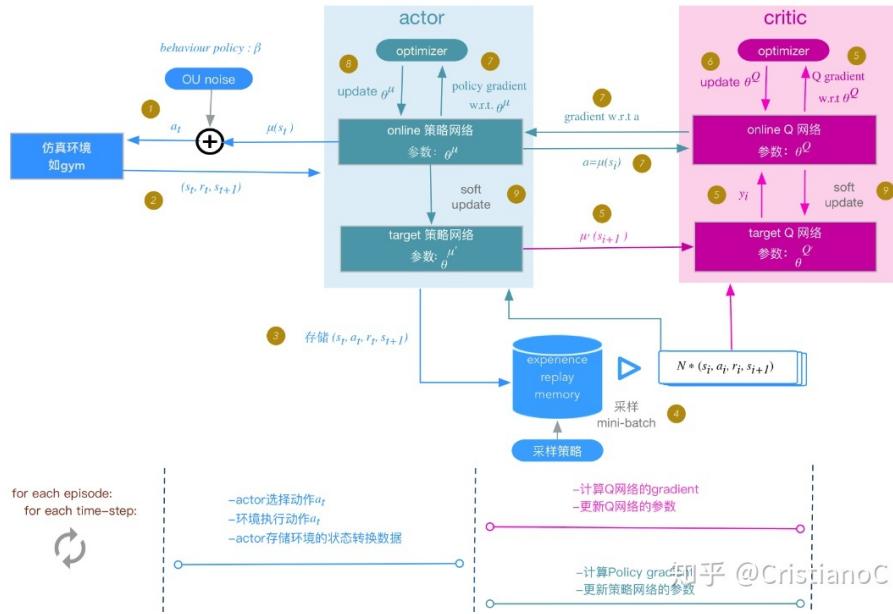
- target 网络参数变化小，用于在训练过程中计算 online 网络的 gradient，比较稳定，训练易于收敛。
- 参数变化小，学习过程变慢。

### 算法流程

1. actor 根据 behavior 策略选择一个  $a_t$ ，下达给 gym 执行该  $a_t$ 。behavior 策略是一个根据当前 online 策略  $\mu$  和随机 UO 噪声生成的随机过程，从这个随机过程采样获得  $a_t$  的值。

$$a_t = \mu(s_t|\theta^\mu) + N_t \quad (6.6)$$

2. gym 执行  $a_t$ ，返回 reward  $r_t$  和新的状态  $s_{t+1}$ ；



3. actor 将这个状态转换过程 (transition):  $(s_t, a_t, r_t, s_{t+1})$  存入 replay memory buffer 中，作为训练 online 网络的数据集。
4. 从 replay memory buffer R 中，随机采样 N 个 transition 数据，作为 online 策略网络、online Q 网络的一个 mini-batch 训练数据。我们用  $(s_t, a_t, r_t, s_{t+1})$  表示 mini-batch 中的单个 transition 数据。
5. 计算 online Q 网络的 gradient:
6. update online Q: 采用 adam optimizer 更新  $\theta^Q$ ;
7. 计算策略网络的 policy gradient;
8. update online 策略网络: 采用 Adam optimizer 更新  $\theta^{\mu}$ ;
9. soft update target 网络  $\mu'$  和  $Q'$

## 6.3 应用

得益于 DDPG 算法能够对连续性信息进行学习，因而其在机器人自主避障、农业机器人自主采摘等方面有着广阔的应用前景。现以 COMPAG 中 “Collision-free

path planning for a guava-harvesting robot based on recurrent deep reinforcement learning”一文具体介绍 DDPG 算法的农业机器人领域的应用。

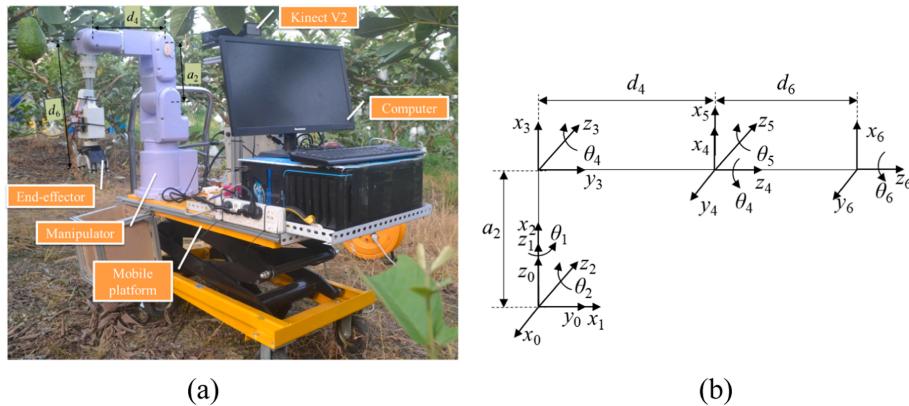


Figure 6.1: (a) 机器人实图 (b) 机械手连路坐标系

1. 学习系统输入信息：植株图像 这一步是为了让机器人“看见”树枝和果实并

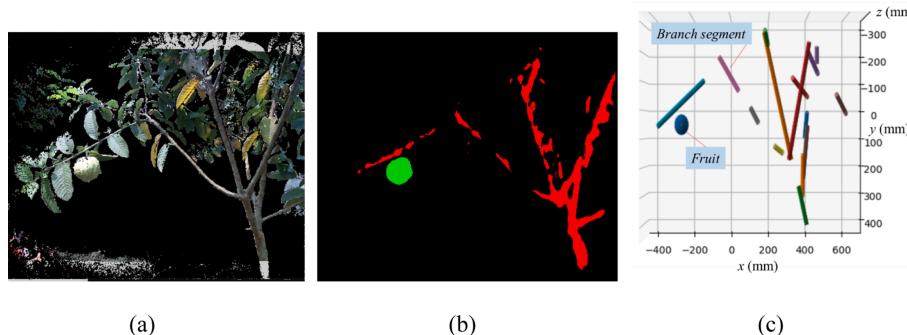


Figure 6.2: (a) 原始图像 (b) 图像分割结果 (c) 图像 3D 建模

作为输出信息，机器人需学习得到避开障碍物、成功抓取果实的最优路径。

2. 建立 DDPG 算法模型，伪代码如下：

3. 训练结果

**Algorithm 1** recurrent DDPG

- 
- 1: Initialize critic network  $Q(s, a|\theta^Q)$  and actor network  $\pi(s|\theta^\pi)$  with weights  $\theta^Q$  and  $\theta^\pi$ , initialize target networks  $Q'(s, a|\theta^{Q'})$  and  $\pi'(s|\theta^{\pi'})$  with weights  $\theta^{Q'}$  and  $\theta^{\pi'}$ , and set replay buffer  $D = \emptyset$
- 2: **For** episode = 1 to M:
- 3: Sample an RGB-D image from the training set, detect 3D fruits and line segments and record their positions as  $D_{fruits}$  and  $D_{lines}$ , and perform a random rigid transform to  $D_{fruits}$  and  $D_{lines}$  to improve randomization
- 4: Randomly select a goal  $x_g$  from  $D_{fruits}$ , set the manipulator to a default pose, and observe a state  $s_0$
- 5: **For** t = 0 to T-1:
- 6: A random action  $a_t$  is selected with probability 25%; otherwise,  $a_t$  is set to  $\pi(s|\theta^\pi)$
- 7: Execute  $a_t$ , receive a reward  $r_t$ , observe a new state  $s_{t+1}$ , and add  $(s_t, a_t, r_t, s_{t+1})$  to D
- 8: Sample N trajectories from D, sample a sub-trajectory with a length from each of these trajectories:  $\{(s_0^n, a_0^n, r_0^n, s_1^n), \dots, (s_{L-1}^n, a_{L-1}^n, r_{L-1}^n, s_L^n)\}_{n=0}^{N-1}$ , and set the initial hidden state of the LSTM of each network for each sub-trajectory to zero
- 9: **For** l = 0 to L-1:
- 10: Calculate  $\{y_i^n = r(s_i^n, a_i^n) + \gamma Q'(s_i^n, \pi'(s_i^n|\theta^{\pi'})|\theta^{Q'})\}_{n=0}^{N-1}$
- 11: Update  $\theta^Q$  by minimizing  $L(\theta^Q) = \sum_{n=0}^{N-1} (y_i^n - Q(s_i^n, a_i^n|\theta^Q))^2$
- 12: Fix  $\theta^Q$  and update  $\theta^\pi$  by maximizing  $L(\theta^\pi) = \sum_{n=0}^{N-1} (Q(s_i^n, \pi(s_i^n|\theta^\pi)|\theta^Q))^2$
- 13: Update the hidden state of the LSTM of each network
- 14: Update the target networks:  $\theta^{\pi'} = \tau\theta^\pi + (1-\tau)\theta^{\pi'}$ ,  $\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}$
- 

Path lengths of recurrent DDPG, DDPG, and bidirectional RRT on test set.

Algorithm	Average (mm)	Standard deviation (mm)
Recurrent DDPG	498.81	123.05
DDPG	770.58	344.83
Bidirectional RRT	585.91	127.88

Running time of recurrent DDPG, DDPG, and bidirectional RRT on test set.

Algorithm	Average (s)	Standard deviation (s)
Recurrent DDPG	0.029	0.030
DDPG	0.040	0.029
Bidirectional RRT	8.005	7.167

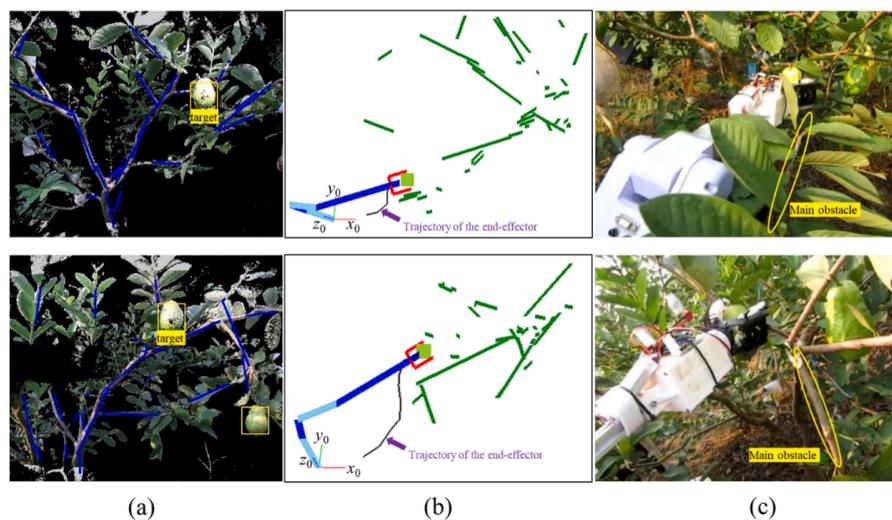


Figure 6.3: 机器人使用循环 DDPG 作为现场路径规划器：(a) 水果和障碍物定位；  
(b) 无碰撞路径规划；(c) 不与障碍物碰撞地接近目标。

# Chapter 7

## 总结和体会

### 7.1 三种决策树算法之比较

在决策树部分，我从基础的 ID3 算法开始讲起，依次介绍了进阶版本的 C4.5 算法模型和 CART 算法模型，现对三种算法进行比较分析。

#### 1. ID3

ID3 算法的基本原理十分简单。该算法的输入训练数据是一组带有类别标记的样本，输出的结果是一棵多叉树，在树中的分支节点上选取能对数据进行最优划分的属性。最优划分的标准是最大信息熵增益。信息熵是信息论里面的概念，是信息的度量方式，不确定度越大或者说越混乱，熵就越大。熵的计算公式：

$$-\sum_{i=1}^n P(i) * \log_2 p(i)^2 \quad (7.1)$$

ID3 决策树可以有多个分支，在决策树的搭建过程中，是按照该特征的所有取值来切分的；一旦按某个特征切分后，该特征在之后的算法执行中，将不再起作用，所以这种切分方式过于迅速。

ID3 不能处理特征值为连续的情况；另外信息增益对可取值数目较多的属性有所偏好，比如通过 ID 号可将每个样本分成一类，但是没有意义；而且 ID3

算法没有剪枝策略，所以生成的树容易产生过拟合，分得太细，考虑条件太多。

## 2. C4.5

C4.5 是对 ID3 的改进，其基本过程与 ID3 类似，改进的地方在于：

- (a) 划分的标准改为信息增益率，信息增益率在信息增益的基础上对属性有一个惩罚，抑制可取值较多的属性，增强泛化性能。
- (b) 可以处理连续型数据，采用先排序后切分的方法，但由于过程中出现多次顺序扫描和排序，导致算法低效。
- (c) 能处理缺失了一些属性的数据。处理的方式通常有 3 种：赋上该属性最常见的值；根据节点的样例上该属性值出现的情况赋一个概率；丢弃有缺失值的样本。
- (d) 利用后置剪枝，合并相邻的无法产生大量信息增益的叶节点，消除过拟合问题。

## 3. CART

分类回归树（Classification and Regression Tree, CART），指的是既可以进行分类又可以进行回归的算法，既可以处理离散属性，也可以处理连续的。两者在建树的过程稍有差异。回归树用最小剩余方差来决定树的最优划分，该划分准则是期望划分之后的子树误差方差最小。分类树使用 Gini 指数来选择划分属性：在所有候选属性中，选择划分后 Gini 指数最小的属性作为优先划分属性。对于一个数据集 T，其 Gini 计算方式为：

$$Gini(T) = 1 - \sum_{i=1}^n p_j^2 \quad (7.2)$$

式中，n 表示类别数， $p_j$  表示数据样本属于类别的概率。

CART 算法总是将当前样本集划分为 2 个子集，即 CART 算法生成的树是一棵二叉树。如果训练数据具有 2 个以上的类别，CART 会通过组合人为地创建二取值序列将目标类别合并成 2 个超类别，这个过程称为双化。

因此 CART 算法适用于样本特征的取值为是或非的场景，数据越复杂，算法的优越性就越显著。而不适用于离散特征有多个可能取值的场景。

参考《决策树算法的比较与应用研究》一文中对不同决策树算法的实验，从准确率、精确率、召回率以及运行时间 4 个方面对各类算法进行评价。其中准确率是最常见的评价指标，指的是被正确分类的样本数占所有样本数的比例，通常来说，正确率越高的分类器越好；精确率指的是在所有分类结果为正的样本在实际为正的比例，度量的是精确性。而召回率是针对原样本而言的，它表示的是样本中正例有多少被预测准确，是覆盖面的度量。结果如下：

数据集 名称	数据集描述
Haberman	Haberman's Survival Data, 该数据集包含 1958 年到 1970 年间在芝加哥大学比林斯医院对所有患有乳腺癌患者所做调查得到的数据，有 306 个样本，4 个属性
Liver	Indian Liver Patient Dataset, 该数据集主要针对印第安肝癌患者记录，数据集分两类，9 个数值属性，1 个非数值属性，具有缺失值，共 583 个样本
Parkinsons	Parkinsons Disease Data Set 该数据集是在患帕金森症病人的生物学噪音测试搜集而来。数据表中每一列是一个特殊的噪音测量项，每一行对应每一个样本相对应的噪音记录，每一个患者有 6 个记录值。数据集共含有 197 个样本，23 个属性
Pima	Pima Indians Diabetes Database, 该数据集记录的是美国某地具有匹马印第安血统的超过 21 岁的女性的医疗记录，用于预测比马印第安人 5 年内糖尿病的发病情况。该数据集含有 768 个样本，分为 2 类，共有 8 个属性
Wdbc	Breast Cancer Wisconsin (Diagnostic), 该数据中的特征是从患者乳房肿块的细针提取物的电子照片中提取的，描述了电子照片中的细胞核的特性。该数据集中含有 569 个样本，32 个属性，分 2 类

	ID3	C4.5	CART
Runtime	0.015	0.015	0.031
Precision	65.52%	64.37%	66.67%
Recall	95.00%	93.33%	85.71%
Accuracy	64.52%	62.37%	63.64%

	ID3	C4.5	CART
Runtime	0.172	0.156	0.031
Precision	71.18%	71.17%	74.73%
Recall	96.80%	92.80%	67.33%
Accuracy	69.71%	68.00%	61.64%

	ID3	C4.5	CART
Runtime	0.202	0.234	0.015
Precision	59.32%	59.32%	87.50%
Recall	100.00%	100.00%	92.11%
Accuracy	59.32%	59.32%	83.67%

	ID3	C4.5	CART
Runtime	0.296	0.297	0.047
Precision	34.36%	34.57%	53.97%
Recall	70.89%	70.89%	49.28%
Accuracy	43.72%	44.16%	66.67%

	ID3	C4.5	CART
Runtime	1.295	1.342	0.016
Precision	77.19%	79.25%	93.10%
Recall	100.00%	95.45%	95.29%
Accuracy	77.19%	77.19%	93.01%

根据测试结果显示，ID3 和 C4.5 在各个数据集中分类效果差不多，是因为本文在 C4.5 的实现中没有考虑到剪枝，且选择的数据集大多属性都为离散的，使得 C4.5 的优势并不明显，另外这 2 个算法表现不稳定，部分数据集分类效果好，部分数据集分类效果不理想。

而对于 CART 算法，在训练时间上要明显优于另外两种，准确率等其他指标也要优于 ID3 和 C4.5。通过以上分析可以看出，3 种决策树分类算法在不同数据集中的表现各有优劣，根据自己的业务需求和侧重点选择不同的算法。在实验中，所有的参数都是用的默认值而没有进行进一步的参数调整以及在数据预处理阶段只是对数据集进行了缺失值的补全并没有进一步处理，这些是导致分类效果都没有达到很好的效果的原因。

## 7.2 弹性网模型

在 10 月份的一次课堂展示中，我讲了关于 LASSO 算法的相关内容，并提到了同时结合 LASSO 模型与 Ridge 模型特征的 Elastic Net 模型（弹性网模型），现利用复盘的机会，查找了相关资料，详细地介绍弹性网模型。

如前文所提到，对于 Ridge 回归：

$$\hat{\beta} = (X^T X + \lambda I)^{-1} X^T Y \quad (7.3)$$

针对岭回归中没有变量选择的问题，1996 年 Robert Tibshirani 提出了 Lasso 回归，因为与岭回归的二次惩罚函数  $\lambda \sum_{i=1}^p \beta_j^2$  相比，相比，Lasso 的一次惩罚函数  $\lambda \sum_{i=1}^p |\beta_j|$  能减小变量系数  $\beta_j$  的收缩程度，所以 Lasso 不仅能把非 0 的预测变量系数  $\beta_j$  向 0 收缩，而且能选择出那些价值较大的预测变量（ $|\beta_j|$  值大的预测变量），从而选出较为准确的模型。同时，Lasso 回归具有较强的稀疏性假设，因此在 Lasso 回归求解过程中，若设计矩阵规模为  $n * p$ ，那么此时最多只能得到  $\min(n, p)$  个变量。尤其当  $p > n$  时，最多只能选出  $n$  个预测变量，所以在这种情况下，Lasso 回归方法就不能选出最真实的模型。另外，当预测变量具有群组效应的时候，利用 Lasso 回归只能选出其中一个变量。

Lasso 回归的目标函数是：

$$\hat{\beta} = \operatorname{argmin}_{\beta} \frac{1}{2} \|Y - X\beta\|_2^2 + \zeta \|\beta\|_1 \quad (7.4)$$

$$J(\beta) = \frac{1}{2} \|Y - X\beta\|_2^2 + \zeta \|\beta\|_1 \quad (7.5)$$

因为我们普通的梯度下降法、最小二乘法、牛顿法针对的情况是损失函数满足可导条件的时候，而 Lasso 回归中的目标函数含有 L1 范数，这种范数的实质是求解绝对值之和，且存在不可导的点。其中一种求解 Lasso 回归稀疏向量的较为普遍的方法是最小角回归算法，另外一种较为常见也是应用较为广泛的方法是 KKT 条件。

KKT 条件是拉格朗日乘子法的拓展，也是一种非常常用的用于解决最优化问题的手段。它的指定作用域包含了不等式，即 KKT 条件是求解带有不等式约束的最优化问题。

假设我们有如下的最优化问题：

$$\min f(x_1, \dots, x_n) \quad (7.6)$$

$$\text{s.t. } \begin{cases} g_i(x_1, \dots, x_n) = 0, i = 1, 2, \dots, k \\ h_i(x_1, \dots, x_n) \leq 0, i = 1, 2, \dots, l \\ x_1, \dots, x_n \geq 0 \end{cases} \quad (7.7)$$

那么该问题的拉格朗日函数为：

$$\begin{aligned} L(x_1, \dots, x_n, \lambda_1, \dots, \lambda_n, \mu_1, \dots, \mu_n) \\ = f(x_1, \dots, x_n) - \sum_{i=1}^k \lambda_i g_i(x_1, \dots, x_n) - \sum_{i=1}^l \mu_i h_i(x_1, \dots, x_n) \end{aligned} \quad (7.8)$$

上述问题的 KKT 条件如下：

$$K.K.T. \begin{cases} \frac{\partial L}{\partial x} = 0, \forall i \quad (\text{Stationarity}) \\ \mu_i * h_i = 0, \forall i \quad (\text{Complementary Slackness}) \\ \lambda_i, \mu_i \geq 0, \forall i \quad (\text{Dual Feasibility}) \\ g_i(x) = 0, \forall i \quad (\text{Primal Feasibility}) \end{cases} \quad (7.9)$$

其中的 KKT 条件为：

$$\begin{cases} X^T(X\hat{\beta} - Y) + \zeta \hat{s} = 0 \\ \hat{s}_i = \text{sign}(\hat{\beta}_j) \quad \text{if} \quad \hat{\beta}_j \neq 0 \\ \hat{s}_i \in [-1, 1] \quad \text{if} \quad \hat{\beta}_j = 0 \end{cases} \quad (7.10)$$

最终求得的解为：

$$\hat{\beta} = (X X^T)^{-1} (X^T Y - \zeta \hat{s}) \quad (7.11)$$

2005 年相关研究者基于岭回归与 Lasso 回归提出了弹性网回归，弹性网回归的惩罚函数为  $\lambda \sum_{i=1}^p [\alpha |\beta_j| + (1 - \alpha) \beta_j^2]$ ，它是 Lasso 回归惩罚函数与岭回归惩罚函数的凸组合，使得它既可以解决具有群组效应的预测变量，又可以像 Lasso 回归那样进行变量选择，得到一个较为简洁准确的模型。

弹性网回归考虑优化问题，得到其估计量为：

$$\hat{\beta} = \operatorname{argmin}_{\beta} \frac{1}{2} \|Y - X\beta\|_2^2 + \lambda \|\beta\|_1 + \frac{\lambda_2}{2} \|\beta\|_2^2 \quad (7.12)$$

$$(7.13)$$

其中  $\lambda_1 > 0, \lambda_2 > 0$  为正则化参数，弹性网不仅具有 L1 正则化的稀疏性，同时兼顾了选择组相关变量的能力，使得尽可能多的数据的重要特征变量被保留，具体可以分析下面的公式。

令  $\alpha = \frac{\lambda_1}{\lambda_1 + \lambda_2/2}, \lambda = \lambda_1 + \lambda_2/2$ ，则上述公式的等价形式为：

$$\hat{\beta} = \operatorname{argmin}_{\beta} \frac{1}{2} \|Y - X\beta\|_2^2 + \lambda [\alpha \|\beta\|_1 + (1 - \alpha) \|\beta\|_2^2] \quad (7.14)$$

$$(7.15)$$

当  $\alpha = 0$  时，弹性网回归变为岭回归，而当  $\alpha = 1$  时，弹性网回归又成为 Lasso 回归。

基于弹性网回归的目标函数，以及求解 Lasso 回归时所用的 KKT 条件，得到求解弹性网回归中的系数估计。弹性网回归的 KKT 条件为：

$$\begin{cases} X^T(X\hat{\beta} - Y) + \lambda_1 \hat{s} + \lambda_2 \hat{\beta} = 0 \\ \hat{s}_i = \operatorname{sign}(\hat{\beta}_j) \quad \text{if } \hat{\beta}_j \neq 0 \\ \hat{s}_i \in [-1, 1] \quad \text{if } \hat{\beta}_j = 0 \end{cases} \quad (7.16)$$

得到系数向量估计过程如下：

$$X^T X \hat{\beta} - X^T Y + \lambda_1 \hat{s} + \lambda_2 \hat{\beta} = 0, \quad (7.17)$$

$$(X^T X + \lambda_2 I_p) \hat{\beta} = X^T Y - \lambda_1 \hat{s} \quad (7.18)$$

最终得到回归系数的估计值为：

$$\hat{\beta} = n^{-1} (S_n + r I_p)^{-1} X^T Y - n^{-1} \lambda_1 (S_n + r I_n)^{-1} \hat{s} \quad (7.19)$$

其中， $r = \frac{\lambda_2}{n}, S_n = n^{-1} X^T X$

## 7.3 心得与总结

从九月夏末至一月寒冬，本学期的《前沿数学专题讨论》课程走过了一段精彩的旅程。首先感谢张南松老师的耐心指导，带领我们领略数学之美，感谢一起上课的七位数院同学，我们一起进步，谢谢你们对我的包容和支持！

这门课是我本科阶段收获最大的一门课，原因有如下几点，一是借用课程的机会，我阅读了许多前沿文献，从科研小白逐步入门；二是这门课让我学会了如何把严谨的理论知识用教学的方式说明白，如何展示；三是通过数次的算法学习，我总结出了自学新算法的一些技巧，如何从零学起一个算法，如何阅读伪代码，如何寻找 demo 资源等等，我相信这些技能将会为我之后的科研生涯提供莫大的助攻。

论语有言，“教学相长也”。非常感谢这学期讨论班给我们每位同学提供的教学机会。事实上，通过本学期的磨练，我更加意识到，只有完整地把知识讲清楚，并让听课者也能掌握这些知识，我才算是完全掌握了。在复盘、推敲知识的过程中，会有许多困惑产生，偶尔也会有灵感乍现，抽丝剥茧，一步一个脚印，这便是学习的过程。本课程的教学方式对于即将深造的本科生来说非常合适。这门本科生课程给我们很好的提供了展示的平台，接触学术前沿的机会。

这学期讨论班所展示的内容主要围绕这段时间的科研工作展开，一项是用 2D-Gabor 滤波对鸡蛋图像进行视觉处理，因为 2D-Gabor 有许多参数需要设置，有一个较大的超参数空间需要优化。因此我学习了 CMA-ES 算法，进行算法全局调优，目前这一工作仍在进行中，希望优化算法能够提高程序的性能。第二项工作主要是对机器人的控制系统进行优化，用机器学习的算法结合强化学习提高控制系统的鲁棒性、适应性和收敛速度。

对于本课程的学习，我认为是一个渐至佳境的过程。在前三次的展示中，我讲了三种非常基础的机器学习算法，决策树算法，EM 算法，SVM 算法。在刚接触这些算法的时候，一方面是经验不足，一方面是信心不够。现在回看当时写的课程讲义，莫名喜感，因为论质量和严谨度，都有许多需要改进的地方。还记得第三次课堂展示，我当时用教室白板推导 EM 算法数学理论，整个过程磕磕绊绊。

随着经验的积累，到后期我面对更为复杂的算法时，也比之前更加从容。对于本学期学习过程中遇到的疑难问题，我罗列如下：

1. 对于 SVM 算法，当时有读几篇关于合理设置 SVM 核函数的前沿论文，但奈何时间限制和数理功底所限，没能完整地进行公式推导。
2. 关于 LASSO 回归参数  $\lambda$  的选择背后的数学原理，我未能透彻掌握。关于参数选择的方法有许多，目前较为主流的有 KKT 方法。在制作算法 demo 的时候，我运用了 R 语言的程序封装包，但未能进行细化的实验，去更加直观理解参数选择的过程。
3. 关于 DDPG 算法，我未能细致的阅读伪代码，在展示的时候将更多的精力花费在算法应用上，自己对于算法的底层逻辑并没有很好的掌握。

总而言之，作为班里唯一一位工科生，我的学习思路和其他数院同学较为不同的一点在于，我会更偏向于算法的应用，我也希望能够将较为前沿的算法用在农业机器人领域。

最后，感谢老师和同学们为我带了这样生动有趣的课程！希望经过讨论班的磨练，今后我在面对科研难题时能够更加坚定从容。

# Chapter 8

## 参考文献

Myles A J, Feudale R N, Liu Y, et al. An introduction to decision tree modeling[J]. Journal of Chemometrics: A Journal of the Chemometrics Society, 2004, 18(6): 275-285.

Safavian S R, Landgrebe D. A survey of decision tree classifier methodology[J]. IEEE transactions on systems, man, and cybernetics, 1991, 21(3): 660-674.

Swain P H, Hauska H. The decision tree classifier: Design and potential[J]. IEEE Transactions on Geoscience Electronics, 1977, 15(3): 142-147.

张海燕, 刘岩, 马丽萌, 苑津莎, 巨汉基, 魏彤珈. 决策树算法的比较与应用研究[J]. 华北电力技术, 2017(06):42-47.DOI:10.16308/j.cnki.issn1003-9171.2017.06.008.

Lin G, Zhu L, Li J, et al. Collision-free path planning for a guava-harvesting robot based on recurrent deep reinforcement learning[J]. Computers and Electronics in Agriculture, 2021, 188: 106350.

Gautron R, Maillard O A, Preux P, et al. Reinforcement learning for crop management support: Review, prospects and challenges[J]. Computers and Electronics in Agriculture, 2022, 200: 107182.

Tang N, Tan X, Cai Y, et al. Characterizations and application potentials of the hemicelluloses in waste oil-tea camellia fruit shells from Southern China[J].

Industrial Crops and Products, 2022, 178: 114551.

Li Y, Guo S, Zhu L, et al. A recurrent reinforcement learning approach applicable to highly uncertain environments[J]. International Journal of Advanced Robotic Systems, 2020, 17(2): 1729881420916258.

Wu D, Zhao E, Fang D, et al. Determination of Vibration Picking Parameters of Camellia oleifera Fruit Based on Acceleration and Strain Response of Branches[J]. Agriculture, 2022, 12(8): 1222.

蒋仕旗, 戴家佳. Logistic 回归模型的一种改进弹性网估计 [J]. 数学理论与应用, 2022, 42(02):108-119.

刘丽红. 基于弹性网的线性模型方差推断及其应用 [D]. 青岛大学, 2021. DOI:10.27262/d.cnki.gqdau.2021.001861.

S. Chen, Z. Fang, S. Lu and C. Gao, "Efficacy of Regularized Multitask Learning Based on SVM Models," in IEEE Transactions on Cybernetics, 2022, doi: 10.1109/TCYB.2022.3196308.

Zou H, Hastie T. Regularization and variable selection via the elastic net[J]. Journal of the royal statistical society: series B (statistical methodology), 2005, 67(2): 301-320.

Li Q, Lin N. The Bayesian elastic net[J]. Bayesian analysis, 2010, 5(1): 151-170.

Jia J, Yu B. On model selection consistency of the elastic net when  $p \approx n$ [J]. Statistica Sinica, 2010: 595-611.

McLachlan G J, Krishnan T. The EM algorithm and extensions[M]. John Wiley Sons, 2007.

Wu C F J. On the convergence properties of the EM algorithm[J]. The Annals of statistics, 1983: 95-103.

Hansen N. The CMA evolution strategy: a comparing review[J]. Towards a new evolutionary computation, 2006: 75-102.

## CHAPTER 8. 参考文献

---

Hansen N. The CMA evolution strategy: A tutorial[J]. arXiv preprint arXiv:1604.00772, 2016.

Hansen N, Müller S D, Koumoutsakos P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)[J]. Evolutionary computation, 2003, 11(1): 1-18.

Daugman J. New methods in iris recognition[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 2007, 37(5): 1167-1175.

Hollingsworth K P, Bowyer K W, Flynn P J. The best bits in an iris code[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2008, 31(6): 964-973.

# Chapter 9

## 附录

### 9.1 附录 1 决策树代码

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction import DictVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz
#1. 获取数据
data = pd.read_csv("F:/数学建模/机器学习/数据/data.csv")
# print(data)
# print(data.describe())
#2. 数据基本处理
#2.1 确定的特征值和目标值
data_x=data[["Pclass","Age","Sex"]]
data_y=data["Survived"]
# print(data_x)
# print(data_y)
# 2.2 确实值得处理
data_x['Age'].fillna(data_x['Age'].mean(), inplace=True) #取平均值
# 2.3 数据集划分
x_train,x_test,y_train,y_test=train_test_split(data_x,data_y,
                                                random_state=22)
```

```
# 3. 特征工程(字典特征抽取)特征中出现类别符号，需要进行one-hot编码处理 (DictVectorizer)
transfer=DictVectorizer(sparse=False)
print(x_train.to_dict(orient="records"))
x_train = transfer.fit_transform(x_train.to_dict(orient="records"))
# 转化为字典类型 x_train.to_dict(orient="records")

x_test = transfer.fit_transform(x_test.to_dict(orient="records"))

# 4. 决策树模型训练和模型评估
estimator=DecisionTreeClassifier()
estimator.fit(x_train, y_train)
# 4.2 评估
print(estimator.score(x_test, y_test))
print(estimator.predict(x_test))
export_graphviz(estimator, out_file="D:/迅雷下载/demo/
                                         machineLearnCode/
                                         decisionTreeTest/tree.dot",
               feature_names=['Age', 'Pclass',
               '女性', '男性'])
```

## 9.2 附录 2 SVM 代码

```

import numpy as np
import matplotlib.pyplot as pl
from sklearn import svm, datasets
from pylab import mpl

mpl.rcParams['font.sans-serif'] = 'SimHei' #画图正常显示中文
mpl.rcParams['axes.unicode_minus'] = False #解决保存图像是负号 '-' 显示方
                                            块的问题

def make_meshgrid(x,y,h=.02):
    """准备用于绘图的网格点
    参数
    -----
    x: x轴数据点
    y: y轴数据点
    h: 间隔距离
    返回值
    -----
    xx, yy: ndarray
    """
    x_min,x_max=x.min()-1,x.max()+1
    y_min,y_max=y.min()-1,y.max()+1
    xx,yy=np.meshgrid(np.arange(x_min,x_max,h),
                      np.arange(y_min,y_max,h))
    return xx,yy

def plot_contours(ax, clf, xx, yy, **params):
    """绘制分类器边界
    参数
    -----
    ax: matplotlib.axes 对象
    xx: 网格点
    yy: 网格点
    params: 控制绘图的其它字典
    """

```

```

Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
out = ax.contour(xx, yy, Z, **params)
return out

# 载入鸢尾花数据集
iris = datasets.load_iris()
# 为了后面方便绘图，这里只使用二个特征
X = iris.data[:, :2]
y = iris.target

C = 1.0
# 备用的各种模型设置
models = (svm.SVC(kernel='linear', C=C),
           svm.LinearSVC(C=C),
           svm.SVC(kernel='rbf', gamma=0.7, C=C),
           svm.SVC(kernel='poly', degree=3, C=C))
# 训练模型
models = (clf.fit(X, y) for clf in models)
# 各模型标题
titles = (u'SVC(线性核)',
           u'LinearsSVC(线性核)',
           u'SVC(RBF)',
           u'SVC(3次多项式核)')

# 把整个图划分成2*2网格
fig, sub = pl.subplots(2, 2, figsize=(12, 8))
pl.subplots_adjust(wspace=0.2, hspace=0.2)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy, alpha=0.8)
    ax.scatter(X0, X1, c=y, cmap=pl.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())

```

```
ax.set_ylim(yy.min(), yy.max())
ax.set_xlabel(u'长')
ax.set_ylabel(u'宽')
ax.set_xticks(()) # 不显示坐标
ax.set_yticks(()) # 不显示坐标
ax.set_title(title)

pl.show()
```

### 9.3 附录 3 EM 代码

```

import math

def cal_mu(w,p,q,xi):
    return w * math.pow(p, xi) * math.pow(1 - p, 1 - xi) / \
           float(w * math.pow(p, xi) * math.pow(1 - p, 1 - xi) +
                  (1 - w) * math.pow(q, xi) * math.pow(1 - q, 1 - xi))

def e_step(w,p,q,x):
    mu=[cal_mu(w,p,q,xi) for xi in x]
    return mu

def m_step(mu,x):
    w=sum(mu)/len(mu)
    p=sum([mu[i]*x[i] for i in range(len(mu))])/sum(mu)
    q=sum([(1-mu[i])*x[i] for i in range(len(mu))])/\
        sum([1-mu[i] for i in range(len(mu))])
    return [w,p,q]

def run(x,w,p,q,maxiteration):
    for i in range(maxiteration):
        mu=e_step(w,p,q,x)
        print(i,[w,p,q])
        if [w,p,q]==m_step(mu,x):
            break
        else:
            [w,p,q]=m_step(mu,x)
        print([w,p,q])
    if __name__=="__main__":
        x = [1, 1, 0, 1, 0, 0, 1, 0, 1, 1]
        [w,p,q]=[0.4,0.6,0.7]
        run(x,w,p,q,100)

```

## 9.4 附录 4 Lasso 代码

```
import copy

def CoordinateDescent(x, y, epochs, learning_rate, Lambda):
    m=x.shape[0]
    X = np.concatenate((np.ones((m,1)),x),axis=1)
    xMat=np.mat(X)
    yMat =np.mat(y.reshape(-1,1))

    w = np.ones(X.shape[1]).reshape(-1,1)

    for n in range(epochs):

        out_w = copy.copy(w)
        for i,item in enumerate(w):
            #在每一个W值上找到使损失函数收敛的点
            for j in range(epochs):
                h = xMat * w
                gradient = xMat[:,i].T * (h - yMat)/m + Lambda * np.sign(w[i])
                w[i] = w[i] - gradient* learning_rate
                if abs(gradient)<1e-3:
                    break
            out_w = np.array(list(map(lambda x:abs(x)<1e-3, out_w-w)))
            if out_w.all():
                break
    return w
```

## 9.5 附录 5 ES 代码

```

import numpy as np
import matplotlib.pyplot as plt

DNA_SIZE = 1           # DNA (real number)
DNA_BOUND = [0, 20]     # solution upper and lower bounds
N_GENERATIONS = 200
POP_SIZE = 100          # population size
N_KID = 50             # n kids per generation

def F(x): return np.sin(10*x)*x + np.cos(2*x)*x      # to find the
                                                       maximum of this function

# find non-zero fitness for selection
def get_fitness(pred): return pred.flatten()

def make_kid(pop, n_kid):
    # generate empty kid holder
    kids = {'DNA': np.empty((n_kid, DNA_SIZE))}
    kids['mut_strength'] = np.empty_like(kids['DNA'])
    for kv, ks in zip(kids['DNA'], kids['mut_strength']):
        # crossover (roughly half p1 and half p2)
        p1, p2 = np.random.choice(np.arange(POP_SIZE), size=2, replace=False)
        cp = np.random.randint(0, 2, DNA_SIZE, dtype=np.bool) # crossover points
        kv[cp] = pop['DNA'][p1, cp]
        kv[~cp] = pop['DNA'][p2, ~cp]
        ks[cp] = pop['mut_strength'][p1, cp]
        ks[~cp] = pop['mut_strength'][p2, ~cp]

    # mutate (change DNA based on normal distribution)
    ks[:] = np.maximum(ks + (np.random.rand(*ks.shape)-0.5), 0.) # must > 0

```

```

        kv += ks * np.random.randn(*kv.shape)
        kv[:] = np.clip(kv, *DNA_BOUND)      # clip the mutated value
    return kids

def kill_bad(pop, kids):
    # put pop and kids together
    for key in ['DNA', 'mut_strength']:
        pop[key] = np.vstack((pop[key], kids[key]))

    fitness = get_fitness(F(pop['DNA']))           # calculate global
                                                    fitness
    idx = np.arange(pop['DNA'].shape[0])
    good_idx = idx[fitness.argsort()][-POP_SIZE:]  # selected by
                                                    fitness ranking (not value)
    for key in ['DNA', 'mut_strength']:
        pop[key] = pop[key][good_idx]
    return pop

pop = dict(DNA=5 * np.random.rand(1, DNA_SIZE).repeat(POP_SIZE, axis=0)
          ,   # initialize the pop DNA values
              mut_strength=np.random.rand(POP_SIZE, DNA_SIZE))
          # initialize the pop
          # mutation strength values

plt.ion()      # something about plotting
x = np.linspace(*DNA_BOUND, 200)
plt.plot(x, F(x))

for _ in range(N_GENERATIONS):
    # something about plotting
    if 'sca' in globals(): sca.remove()
    sca = plt.scatter(pop['DNA'], F(pop['DNA']), s=200, lw=0, c='red',
                      alpha=0.5); plt.pause(0.05)

# ES part

```

```
    kids = make_kid(pop, N_KID)
    pop = kill_bad(pop, kids)    # keep some good parent for elitism

plt.ioff(); plt.show()
```

## 9.6 附录 6 CMA-ES 代码

```

import numpy as np
import numpy.linalg as la

from matplotlib import cm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import cma

np.random.seed(0)

"""

The Algorithm

Parameters
Population size : N
Covariance Matrix: C
mean of best k : mu

1. Sample N from multivariate normal distribution
2. Calculate fitness and

"""

def rastrigin(X, A=10):
    return A + np.sum((X**2 - A * np.cos(2 * np.pi * X)), -1)

def rosenbrock(X, a=1, b=100):
    x, y = X[:, 0], X[:, 1]
    return (a-x)**2 + b*(y-x**2)**2

#(1-x)^2+100(y-x^2)^2

def _plot_points(X, low=-10, high=10, size=1024):

```

```

""" Plots the points on same scale as image """
tmp = (X - low)/(high-low) * size
plt.scatter(tmp[:,0], tmp[:,1], color='black', alpha=.5)

if __name__ == '__main__':
    low, high, size = -6, 6, 2048
    spacing = np.linspace(low, high, size)

    grid = np.stack(np.meshgrid(spacing, spacing), -1)
    function = lambda X: rastrigin(X + [3,2])      # Move the min away
                                         from (0,0)
#function = rosenbrock
Z = function(grid)

n = 200      # Population size
d = 2        # Dimensions
k = 50       # Size of elite population

X = np.random.normal(0,1.24, (d, n))

for i in range(24):
    # Minimize this function
    fitness = function(X.T)
    arg_topk = np.argsort(fitness)[:k]
    topk = X[:,arg_topk]

    #print(f'Iter {i}, score {fitness[arg_topk[0]]}, X = {X[:, arg_topk[0]]}')
    # Covariance of topk but using mean of entire population
    centered = topk - X.mean(1, keepdims=True)
    C = (centered @ centered.T)/(k-1)
    # Eigenvalue decomposition
    w, E = la.eigh(C)
    # Generate new population
    # Sample from multivariate gaussian with mean of topk

```

```
N = np.random.normal(size=(d,n))
X = topk.mean(1,keepdims=True) + (E @ np.diag(np.sqrt(w)) @ N)
if i % 1 == 0:
    print(f'iter {i}, z= {fitness[arg_topk[0]]:.2f}, x= {X[:, arg_topk[0]].round(2)}')
plt.clf()
plt.imshow(Z, cmap='Oranges')
plot_points(X.T, low, high, size)
plt.pause(.2)
plt.draw()
plt.savefig(f'plots/fig-{i}.jpg')
```

## 9.7 附录 7 DDPG 代码

```

def prepare_params(kwargs):
    # DDPG params
    ddpg_params = dict()

    env_name = kwargs['env_name']

    def make_env():
        return gym.make(env_name)
    kwargs['make_env'] = make_env

    tmp_env = cached_make_env(kwargs['make_env'])
    assert hasattr(tmp_env, '_max_episode_steps')
    kwargs['T'] = tmp_env._max_episode_steps
    tmp_env.reset()
    kwargs['max_u'] = np.array(kwargs['max_u']) if isinstance(kwargs['
                                                max_u'], list) else kwargs['
                                                max_u']
    kwargs['gamma'] = 1. - 1. / kwargs['T']

    if 'lr' in kwargs:
        kwargs['pi_lr'] = kwargs['lr']
        kwargs['Q_lr'] = kwargs['lr']
        del kwargs['lr']

    for name in ['buffer_size', 'hidden', 'layers',
                 'network_class',
                 'polyak',
                 'batch_size', 'Q_lr', 'pi_lr',
                 'norm_eps', 'norm_clip', 'max_u',
                 'action_l2', 'clip_obs', 'scope', 'relative_goals']:
        ddpg_params[name] = kwargs[name]
        kwargs['_' + name] = kwargs[name]
        del kwargs[name]

    kwargs['ddpg_params'] = ddpg_params

    return kwargs

def configure_ddpg(dims, params, reuse=False, use_mpi=True,
                  clip_return=True):

```

```

sample_her_transitions = configure_her(params)
# Extract relevant parameters.
gamma = params['gamma']
rollout_batch_size = params['rollout_batch_size']
ddpg_params = params['ddpg_params']

input_dims = dims.copy()

# DDPG agent
env = cached_make_env(params['make_env'])
env.reset()
ddpg_params.update({'input_dims': input_dims, # agent takes an
                     input observations
                     'T': params['T'],
                     'clip_pos_returns': True, # clip positive
                                             returns
                     'clip_return': (1. / (1. - gamma)) if
                                     clip_return
                                             else np.
                                             inf, # max
                                             abs of
                                             return
                     'rollout_batch_size': rollout_batch_size,
                     'subtract_goals': simple_goal_subtract,
                     'sample_transitions': sample_her_transitions,
                     'gamma': gamma,
                     'bc_loss': params['bc_loss'],
                     'q_filter': params['q_filter'],
                     'num_demo': params['num_demo'],
                     })
ddpg_params['info'] = {
    'env_name': params['env_name'],
}
policy = DDPG(reuse=reuse, **ddpg_params, use_mpi=use_mpi)
return policy

def main():
    experiment= 'InvertedPendulum-v1' #specify environments here

```

```

env= gym.make(experiment)
steps= env.spec.timestep_limit #steps per episode
assert isinstance(env.observation_space, Box), "observation space
                                              must be continuous"
assert isinstance(env.action_space, Box), "action space must be
                                              continuous"

#Randomly initialize critic,actor,target critic, target actor
#network and replay buffer
agent = DDPG(env, is_batch_norm)
exploration_noise = OUNoise(env.action_space.shape[0])
counter=0
reward_per_episode = 0
total_reward=0
num_states = env.observation_space.shape[0]
num_actions = env.action_space.shape[0]
print "Number of States:", num_states
print "Number of Actions:", num_actions
print "Number of Steps per episode:", steps
#saving reward:
reward_st = np.array([0])

for i in xrange(episodes):
    print "==== Starting episode no:",i,"====","\n"
    observation = env.reset()
    reward_per_episode = 0
    for t in xrange(steps):
        #rendering environmet (optional)
        env.render()
        x = observation
        action = agent.evaluate_actor(np.reshape(x,[1,num_states]))
        noise = exploration_noise.noise()
        action = action[0] + noise #Select action according to
                                  #current policy and
                                  #exploration noise
        print "Action at step", t ,":",action,"\n"

```

```
observation,reward,done,info=env.step(action)

#add s_t,s_t+1,action,reward to experience memory
agent.add_experience(x,observation,action,reward,done)
#train critic and actor network
if counter > 64:
    agent.train()
reward_per_episode+=reward
counter+=1
#check if episode ends:
if (done or (t == steps-1)):
    print 'EPISODE: ',i,' Steps: ',t,' Total Reward: ',
          reward_per_episode
    print "Printing reward to file"
    exploration_noise.reset() #reinitializing random noise
                                for action
                                exploration
    reward_st = np.append(reward_st,reward_per_episode)
    np.savetxt('episode_reward.txt',reward_st, newline='\n')
print '\n\n'
break
total_reward+=reward_per_episode
print "Average reward per episode {}".format(total_reward /
                                              episodes)
```