《机器学习与 Python 实践》习题答案

第一版

August 11, 2021

Contents

1 机器学习概述

1.1 查阅维基百科中相关概念，如数据科学、机器学习、数据挖掘、人工智能、统计学习的定义，并进行对比。

答: Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains.

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.

Data mining is a process of extracting and discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems. Data mining is an interdisciplinary subfield of computer science and statistics with an overall goal to extract information (with intelligent methods) from a data set and transform the information into a comprehensible structure for further use.

Artificial intelligence (AI) is intelligence demonstrated by machines, as opposed to the natural intelligence displayed by humans or animals. Leading AI textbooks define the field as the study of "intelligent agents": any system that perceives its environment and takes actions that maximize its chance of achieving its goals.

Statistical learning theory is a framework for machine learning drawing from the fields of statistics and functional analysis. Statistical learning theory deals with the problem of finding a predictive function based on data.

1.2 查阅最近流行的 3-5 个机器学习方法，并填入表 1.1 中。

表 1: 机器学习方法

	无监督学习	有监督学习	强化学习
浅层学习	自组织映射		
深度学习	BERT	Inception、Transformer	SAC、TD3

1.3 什么原因导致了深度学习的盛行？

答: 主要原因包括数据量的高速增长、算法创新与神经网络训练方法改进、计算机运算性能提升以及开源深度学习框架的完善等

1.4 谷歌翻译使用了哪种机器学习方法？

答: 谷歌翻译算法采用了 Seq2seq 模型,模型的编码器和解码器之间有一个 8 层的 LSTM-RNN 结构，这种结构采用了部分连接的方式来改善模型的运行速度和效果。其中，LSTM(Long Short-Term Memory networks) 是一种特殊的循环神经网络（RNN），能够捕捉和学习到长序列中的相关性。基础的 Seq2Seq(Sequence to Sequence) 模型由两个 RNN 模型组成，一个用于对输入系列进行编码，一个用于对输出序列进行解码。目前的谷歌翻译算法采用了一个统一的翻译系统，取代了数以万计的翻译子系统。

1.5 除了棋类游戏，目前人工智能还在哪些领域超过了人类的表现？

答: 知识检索领域、智力问答、游戏、电子竞技、魔方、拼字游戏、网络安全等。

1.6 统计学的极大似然估计可以应用到表 1.1 中的哪些方法里？

答: 混合模型、PCA、线性回归、逻辑回归

2 Python 科学计算简介

2.1 使用列表推荐 (list comprehension)，生成 20 以内的乘法表。

```
result = [str(j)+'*'+str(i)+'='+str(i*j) for i in range(1,21) for
    j in range(1,21) if i>=j]
print(result)
```

2.2 仅使用乘法，写一个求幂函数 power(x,n)，n 为正整数。

```
def power(x,n):
    product = 1
    for i in range(1,n+1):
        product *= x
    return product
```

2.3 写一个函数，输入为一个列表，输出为该列表中所有正数之和。

```
def sumofpositive(s):
    l = len(s)
    sum = 0
    for i in range(0,l):
        if s[i]>0:
            sum += s[i]
    return sum

s = [1,-2,4,-6,3,-2,13]
sumofpositive(s)
```

2.4 产生二元均匀分布随机数，通过计算落入单位圆内的点的比例，估计圆周率。

```
import numpy as np
n = 1000000
arr = np.random.random(size=(n,2))
sum = 0
for i in range(0,n):
    if arr[i][0]**2+arr[i][1]**2<=1:
        sum += 1
pi = 4*sum/n
print(pi)
```

2.5 分别产生 100 个服从正态分布、t 分布、卡方分布的随机数，并画出直方图。

```
import matplotlib.pyplot as plt
import numpy as np

random1 = np.random.randn(100) #正态分布随机数
plt.hist(random1,30,alpha=0.5)
plt.title("正态分布随机数直方图")
plt.show()

random2 = np.random.standard_t(1,100) #t 分布随机数
plt.hist(random2,30,alpha=0.5)
plt.title("t 分布随机数直方图")
plt.show()

random3 = np.random.chisquare(1,100) #卡方分布随机数
plt.hist(random3,30,alpha=0.5)
plt.title("卡方分布随机数直方图")
plt.show()
```

2.6 使用不同的颜色和线条，在同一张图的区间 $[-4\pi, 4\pi]$ 中画出 sin() 函数和 cos() 函数。使用 subplot() 函数将 sin() 函数和 cos() 函数分别画在两张子图中。

```
import matplotlib.pyplot as plt
import numpy as np
import math

x = np.arange(-4*math.pi,4*math.pi,0.1)
y1 = [math.sin(t) for t in x]
y2 = [math.cos(t) for t in x]

#在同一张图的区间中画图
plt.plot(x,y1) #正弦
plt.plot(x,y2) #余弦
plt.show()

#分别画在两张子图中
plt.figure()
plt.subplot(1,2,1) #正弦
plt.plot(x,y1
plt.title('y=sin(x)')
```

```
plt.subplot(1,2,2) #余弦
plt.plot(x,y2)
plt.title('y=cos(x)')
plt.show()
```

2.7 产生一个大小为 3×5 正态分布的随机数矩阵 A，计算 A^TA 和 AA^T，并分别求行列式、逆矩阵、特征值、特征向量。

```
import numpy as np
import numpy.linalg as la
# 矩阵A
A = np.random.random((3,5))
arr1 = np.dot(A,A.T)
arr2 = np.dot(A.T,A)
#arr1
a1 = la.det(arr1)      #行列式
arr1_T = la.inv(arr1) #逆矩阵
u1,v1 = la.eig(arr1)  #特征值和特征向量
#arr2
a2 = la.det(arr2)      #行列式
arr2_T = la.inv(arr2) #逆矩阵
u2,v2 = la.eig(arr2)  #特征值和特征向量
```

2.8 在例 2.2 的 process_stock 类中，

增加一个函数实现如下功能：输入单个日期，输出该日的高-开-低-收价格；

增加一个函数实现如下功能：输入日期列表，输出列表日期中每一天的高-开-低-收价格。

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

class process_stock():
    ##--data and variables--
    stock_name=''
    stock_path=''
    stock_code=''
    stock_quate={}
    stock_return=[]
    date1=[]
```

```
    date2=[]

    ##--functions and methods--
    def read_data1(self):  # 输出该日的高开低收价格
        path=self.stock_path+'/'+self.stock_code
        index0=pd.read_table(path,encoding='cp936',header=None,engine='python')
        index0.columns=['date','o','h','l','c','v','to']
        index1=index0[:-1]
        index1.index=index1['date']
        da1=str(self.date1)
        v1=list(index1.loc[da1,['o']])
        v2=list(index1.loc[da1,['h']])
        v3=list(index1.loc[da1,['l']])
        v4=list(index1.loc[da1,['c']])
        print(da1,'的开盘价为',v1)
        print(da1,'的最高价为',v2)
        print(da1,'的最低价为',v3)
        print(da1,'的成交价为',v4)

    def read_data2(self):   # 输出日期列表中每一天的高开低收价格
        path=self.stock_path+'/'+self.stock_code
        index0=pd.read_table(path,encoding='cp936',header=None,engine='python')
        index0.columns=['date','o','h','l','c','v','to']
        index1=index0[:-1]
        index1.index=index1['date']
        l=len(self.date2)
        for i in range(0,l):
            da=str(self.date2[i])
            v1=list(index1.loc[da,['o']])
            v2=list(index1.loc[da,['h']])
            v3=list(index1.loc[da,['l']])
            v4=list(index1.loc[da,['c']])
            print(da,'的开盘价为',v1)
            print(da,'的最高价为',v2)
            print(da,'的最低价为',v3)
            print(da,'的成交价为',v4)

p = process_stock()
p.stock_path='D:/hs300'
p.stock_code='SZ399300.TXT'
p.date1=20100602
```

```
p.date2=[20100602,20110329,20151201]
p.read_data1()
p.read_data2()
```

2.9 设计一个用于回归分析的类，说明类中数据、变量、和各个函数的功能。

```
import numpy as np
import pandas as pd
from sklearn.linear_model import BayesianRidge, LinearRegression,
    ElasticNet
from sklearn.svm import SVR
from sklearn.ensemble.gradient_boosting import
    GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import explained_variance_score,
    mean_absolute_error, mean_squared_error, r2_score
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.formula.api import ols
import scipy.stats as stats
from pylab import *
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import
    variance_inflation_factor

class regression():
    ##--data and variable--
    data=[]

    ##--function and methods--
    #1. 读入数据与描述性统计分析
    def read_data(self):
        print(self.data)
        #数据描述
        print('数据的描述性统计分析')
        print(self.data.describe())      #描述性统计分析
        self.data.boxplot()              #箱线图
        plt.show()
        print('数据的相关系数矩阵')
        print(self.data.corr())
```

```
    #2. 建立线性回归模型
    def model_building(self):
        df=pd.DataFrame(self.data)
        y=df.iloc[:,0]  #取第一列
        X=df.iloc[:,1:] #取第二列到最后一列
        lm = ols('y ~ X', data=df).fit()
        print(lm.summary())
        #对线性回归进行预测
        model = LinearRegression()
        model = model.fit(X, y)
        Y_pred = model.predict(X)
        print('Y的预测值为',Y_pred)
        #预测情况对比
        print('预测值与实际值的对比图')
        plt.figure()
        plt.plot(range(len(Y_pred)),Y_pred,'b',label="predict")
        plt.plot(range(len(Y_pred)),y,'r',label="Y")
        plt.legend(loc="upper right") #显示图中的标签
        plt.show()

    #3. 模型的检验
    def model_testing(self):
        df=pd.DataFrame(self.data)
        y=df.iloc[:,0]  #取第一列
        X=df.iloc[:,1:]  #取第二列到最后一列
        #正态性检验
        print('正态性检验')
        mpl.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False
        sns.distplot(a = y,
        bins =10,
        fit = stats.norm,
        norm_hist = True,
        hist_kws = {'color':'green','edgecolor':'black'},
        kde_kws =
            {'color':'black','linestyle':'--','label':'核密度曲线'},
        fit_kws =
            {'color':'red','linestyle':':','label':'正态密度曲线'}
        )
        plt.legend()
        plt.show()
```

```
#pp 图和qq 图
pq_plot = sm.ProbPlot(y)
pq_plot.ppplot(line='45')
plt.title('PP 图')
pq_plot.qqplot(line='q')
plt.title('QQ 图')
plt.show()
#多重共线性检验
print('多重共线性检验')
X1 = sm.add_constant(X)
vif = pd.DataFrame()
vif['features'] = X1.columns
vif["VIF Factor"] =
    [variance_inflation_factor(X1.values,i) for i in
    range(X1.shape[1])]
print(vif)
#线性相关性检验 （各变量之间线性相关性检测）
print('线性相关性检验')
sns.pairplot(X) #去除哑变量
plt.show()
#异常值检验
model1=sm.formula.ols('y~X ',data = df).fit()
print('异常值检测')
outliers = model1.get_influence()
print(outliers)
#帽子矩阵
print('帽子矩阵')
leverage = outliers.hat_matrix_diag
print(leverage)
#dffits 值
print('dffits 值')
dffits = outliers.dffits[0]
print(dffits)
#学生化残差
print('学生化残差')
resid_stu = outliers.resid_studentized_external
print(resid_stu)
#cook 距离
print('库克距离')
cook = outliers.cooks_distance[0]
print(cook)
```

```
        # 合并各种异常值检验的统计量值
        contatl = pd.concat([pd.Series(leverage, name =
            'leverage'),
        pd.Series(dffits, name = 'dffits'),
        pd.Series(resid_stu, name = 'resid_stu'),
        pd.Series(cook, name = 'cook')
        ],axis =1 )
        print(contatl.head())

#实证
#P.data 为导入已有数据或生成的数据矩阵，其中第一列为y ，后几列为X
    。
P=regression()
P.data=pd.read_excel(path)
P.read_data()
P.build_model()
P.test_model()
```

2.10 摄氏度和华氏度的转换关系为 $C=(F-32)\times\frac{5}{9}$。（绝对零度为 $-273.15°C$。）

（1）编写一个函数将摄氏度转为华氏度。

（2）编写一个函数，将华氏度转为摄氏度。

（3）编写一个函数，输入为度数和类型（华氏、摄氏），输出为转换后的度数和类型。

（4）使用面向对象的编程来实现以上功能。

```
class temperature_conversion():
    ##--data and variable
    C=[]
    F=[]
    type=''

    ##--function and method
    #1. 将摄氏度转化为华氏度
    def C_to_F(self):
        if self.C>=-273.15:
            F1=9*self.C/5+32
            print('此时的华氏度为',F1)
        else:
            print("Wrong!")

    #2. 将华氏度转化为摄氏度
```

```
    def F_to_C(self):
        if self.F>=-459.67:
            C1=(self.F-32)*5/9
            print('此时的摄氏度为',C1)
        else:
            print("Wrong!")

    #3. 二者相互转化
    def F_C(self):
        if self.type=='摄氏度':
            if self.C>=-273.15:
                F1=9*self.C/5+32
                print('此时的华氏度为',F1)
            else:
                print("Wrong!")
        else:
            if self.F>=-459.67:
                C1=(self.F-32)*5/9
                print('此时的摄氏度为',C1)
            else:
                print("Wrong!")

P1 = temperature_conversion()
P1.C = 39
P1.F = 100.1
P1.type = '摄氏度'
print(P1.C_to_F())
print(P1.F_to_C())
print(P1.F_C())
```

3 无监督学习

3.1 证明 $mean(x) = \arg\min_\mu \sum_{i=1}^{N}(x_i - \mu)^2$，$median(x) = \arg\min_\mu \sum_{i=1}^{N}|x_i - \mu|$。

均值部分：证明可通过求导并将导数设为 0 得到。

中位数部分:假如样本为单数,将样本从小到大重新排序记为 $\{x_1, x_2, ...x_m, x_{m+1}..., x_{2m}, x_{2m+1}\}$，有 $x_1 \leq ... \leq x_{m+1} \leq ... \leq x_{2m+1}$，则有

$$min_\mu \sum_{i=1}^{2m+1}|x_i - \mu| = min_\mu \left[\sum_{i=1}^{m}(|x_i - \mu| + |x_{2m+2-i} - \mu|) + |x_{m+1} - \mu|\right]$$

对于第一部分括号 () 里的内容，每一项取最小值的条件是 μ 在两个样本的中间，则 2m+1 项共同取最小应满足条件 $x_m \leq \mu \leq x_{m+2}$。对于第二部分 $|x_{m+1} - \mu|$，取最小值为 x_{m+1}，在 $x_m \leq \mu \leq x_{m+2}$ 的范围内。综上，式子取最小值时有 $\mu = x_{m+1} = median(x)$.

假如样本为双数，将样本从小到大重新排序记为 $\{x_1, x_2, ...x_m, x_{m+1}..., x_{2m}\}$，有 $x_1 \leq ... \leq x_{m+1} \leq ... \leq x_{2m}$，则有

$$min_\mu \sum_{i=1}^{2m}|x_i - \mu| = min_\mu \left[\sum_{i=1}^{m}(|x_i - \mu| + |x_{2m+1-i} - \mu|)\right]$$

对于括号 () 里的内容，每一项取最小值的条件是 μ 在两个样本的中间，则 2m+1 项共同取最小应满足条件 $x_m \leq \mu \leq x_{m+1}$。

中位数也满足上面的范围，有 $\mu = \frac{x_m + x_{m+1}}{2} = median(x)$.

3.2 在核密度估计程序分析中，改变窗宽大小并观察输出结果。

```
# 核密度估计函数
def ourkde(x,h,u):
    fu = []
    for u0 in u:
        t = (x-u0)/h
        K = h**(-1)*np.exp(-0.5*t**2)/np.sqrt(2*np.pi)
        fu.append(np.mean(K))
        fu = np.asarray(fu)
    return fu
```

```
# 核密度估计应用
x = hs300_ret.values    #见例3.1
u = np.linspace(-0.087,0.067,100)
# 改变窗宽h 的值观察输出结果（接近最优窗宽，结果平滑且合理）
h = 0.005
fu = ourkde(x,h,u)
plt.hist(x,30,normed=True)
plt.plot(u,fu,'-')
# 改变窗宽h 的值观察输出结果（窗宽过小，结果将不平滑，产生过拟合）
h = 0.0005
fu = ourkde(x,h,u)
plt.hist(x,30,normed=True)
plt.plot(u,fu,'-')
# 改变窗宽h 的值观察输出结果（窗宽过大，结果平滑，但效果不好）
h = 0.05
fu = ourkde(x,h,u)
plt.hist(x,30,normed=True)
plt.plot(u,fu,'-')
```

3.3 在 3.1.2 节案例分析中，将标准化后的直方图、正态分布近似估计和核密度估计画在一张图中。

```
# 数据（见案例3.1.2）
x = hs300_ret.values  # 数据
u = np.linspace(-0.087,0.067,100)  #绘图范围
# 正态分布近似估计
miu = np.mean(x)
sigma = np.std(x)
y = np.exp(-(u-miu)**2/(2*sigma**2))/(np.sqrt(2*np.pi)*sigma)
# 核估计(见例3.1)
h =  1.06*np.std(x)*1340**(-0.2)
fu = ourkde(x,h,u)
# 绘图
plt.hist(x,30,normed=True)
plt.plot(u,y,'-',color='blue',label='normal')
plt.plot(u,fu,'-',color='red',label='kernel')
plt.legend()
```

3.4 已知待估计的（一维）密度函数关于 μ 对称，如何得到对称的核密度估计？

答:

$$\hat{f}_h(x) = \frac{1}{2hN}\sum_{i=1}^{N}\left[K\left(\frac{x_i - x + \mu}{h}\right) + K\left(\frac{x_i + x - \mu}{h}\right)\right]$$

3.5 将一维的核密度估计方法推广到二维，并写出二维核密度估计的计算公式和 Python 程序。

答: 假设 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$ 是独立同分布的二维数据样本，其中 $\mathbf{x}_i = (x_{i1}, x_{i2})$，服从概率密度函数 $f(\mathbf{u})$，则核密度估计为：

$$\hat{f}_h(\mathbf{u}) = \frac{1}{N}\sum_{i=1}^{N}K_h(\mathbf{x}_i - \mathbf{u}) = \frac{1}{Nh^2}\sum_{i=1}^{N}K\left(\frac{x_{i1} - u}{h}\right)K\left(\frac{x_{i2} - u}{h}\right)$$

```
# 二维核密度估计函数（高斯核）
def kde2(x,h,u):
    fu = []
    for i in range(np.size(u,1)):
        for j in range(np.size(u,1)):
            t0 = (x[:,0]-u[0][i])/h
            t1 = (x[:,1]-u[1][j])/h
            K =
                h**(-2)*np.exp(-0.5*t0**2)*np.exp(-0.5*t1**2)/2*np.pi
            fu.append(np.mean(K))
    fu = np.asarray(fu)
    return fu

# 生成数据
N = 100
mean = (0, 0)
cov = [[1, 0], [0, 1]]
x = np.random.multivariate_normal(mean,cov,N)
# 估计
u1 = np.linspace(-2.5,2.5,50)
u2 = np.linspace(-2.5,2.5,50)
u = np.vstack([u1,u2])
h = 0.4
fu = kde2(x,h,u)
# 绘图
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
fig = plt.figure()
ax = fig.gca(projection='3d')
uu1 = np.repeat(u1,50)
uu2 = np.tile(u2,50)
ax.plot(uu1,uu2,fu)
```

3.6 证明 K 均值算法迭代中，目标函数取值在每步都减少或不会增大。

答: 一共有 N 个样本点，记为 $\mathbf{x} = (x_1, ..., x_N)$。参考 3.3 节的迭代步骤。

第 t 次迭代时：第 j 个样本点到第 k 类中心点的距离记为 $d^{(t)}(j,k) = ||x_j - \mu_k^{(t)}||$, 第 j 个样本点的类别更新为 $z_k^{(t)} = argmin_{k=1,2,...,K}\left\{d^{(t)}(j,k)\right\}$, 更新属于第 k 类的所有样本均值为新的样本中心点 $\mu_k^{(t+1)}$。此时损失函数为：

$$L^{(t)}(C_1, ...C_K, \mu_1, ..., \mu_K) = \sum_{k=1}^{K}\sum_{i \in C_k} ||x_i - \mu_k^{(t+1)}||^2 = \sum_{j=1}^{N} ||x_j - \mu_{z_k^{(t)}}^{(t+1)}||^2$$

我们证明如下不等式，即第 t+1 次的损失函数是小于等于第 t 次损失函数的：

$$L^{(t+1)} = \sum_{j=1}^{N} ||x_j - \mu_{z_k^{(t+1)}}^{(t+2)}||^2 \leq \sum_{j=1}^{N} ||x_j - \mu_{z_k^{(t+1)}}^{(t+1)}||^2 \leq \sum_{j=1}^{N} ||x_j - \mu_{z_k^{(t)}}^{(t+1)}||^2 = L^{(t)} \quad (1)$$

对于第二个不等式，其中每一项可由下面的定义证明，第 j 个样本新分配的类别使类别中心到样本的距离最小：

$$z_k^{(t+1)} = argmin_{k=1,2,...,K}\left\{d^{(t+1)}(j,k)\right\}$$

则有：

$$d^{(t+1)}(j, z_k^{(t+1)}) = ||x_j - \mu_{z_k^{(t+1)}}^{(t+1)}||^2 \leq ||x_j - \mu_{z_k^{(t)}}^{(t+1)}||^2 = d^{(t+1)}(j, z_k^{(t)})$$

因此有:

$$\sum_{j=1}^{N} ||x_j - \mu_{z_k^{(t+1)}}^{(t+1)}||^2 \leq \sum_{j=1}^{N} ||x_j - \mu_{z_k^{(t)}}^{(t+1)}||^2$$

对于第一个不等式，对每个类别中的样本，可以证明，中心点距离所有样本的平方和是最近的。假设第 k 类样本中有 n_k 个样本点，则有：

$$\frac{\partial \sum_{i=1}^{n_k} ||x_i - \mu||^2}{\partial \mu} = -2\sum_{i=1}^{n_k}(x_i - \mu) = 0 \quad when \quad \mu = \frac{\sum_{i=1}^{n_k} x_i}{n_k}$$

函数是凸的，则取样本点均值时平方和最小，有：

$$\sum_{k=1}^{K}\sum_{i\in C_k}||x_i-\mu_k^{(t+2)}||^2=\sum_{j=1}^{N}||x_j-\mu_{z_k^{(t+1)}}^{(t+2)}||^2\leq\sum_{j=1}^{N}||x_j-\mu_{z_k^{(t+1)}}^{(t+1)}||^2$$

综上，公式 (1) 成立。

3.7 修改 K 均值算法程序，记录每步迭代的目标函数取值并作图观察。

```
def ourkmean(x,k,mu,tol):
    n,p = x.shape
    dist_matx = np.zeros((n,k))
    id = []
    VAL_list = []  #存储目标函数
    iter = 0
    max_it = 100
    diff = 100
    VAL2 = 10000
    while diff>tol and iter<max_it:
        for i in range(k):
            dist_matx[:,i] = np.sum((x-mu[i,:])**2,axis=1)
        id = np.argmin(dist_matx,axis=1)
        VAL=0
        for i in range(k):
            mu[i,:] = np.mean(x[id==i,:],axis=0)
            VAL = VAL+np.sum((x[id==i,:]-mu[i,:])**2)  #目标函数
        VAL_list.append(VAL)  #储存目标函数序列
        diff = np.abs(VAL-VAL2)
        VAL2 = VAL
        iter = iter+1
    return id,mu,VAL_list

# 生成数据
n = 100
x1 = np.random.randn(n,2)
x2 = np.random.randn(n,2) + np.array([2,2])
x = np.vstack((x1,x2))
tol = 0.001
k = 2
mu=np.array([[-0.1,-0.1],[1.0,1.0]])
id,mu,VAL_list = ourkmean(x,k,mu,tol)
#目标函数作图
```

```
plt.figure()
plt.plot(VAL_list)
plt.show()
```

3.8 使用主成分分析方法，分析手写数据集"zip.train" 中其它数字（非数字 3）的特征。

答: 在案例 3.4.3 中将 id3 = data[:,0]==3 中的 3 改为其他数字即可，其他过程仿照案例。

3.9 在 3.4.5 节案例分析的程序中，解释为何 sector1 和 setor1v 的结果很接近但却不完全一致。

答: sector1v 的计算过程是对转置后的数据矩阵得到的 1339×1339 的协方差矩阵进行特征分解，第一主成分为第一个因子在 1339 天交易日上的收益率，收益率矩阵减去每日的市场平均收益后与第一主成分相乘得到第一主成分上的投影得分，相当于去掉市场平均后在第一个因子上个股的因子暴露。对其进行排序得到 sector1v。

sector1 是由 300×300 的协方差矩阵进行特征分解得到的第二主成分方向元素值的排序结果，按奇异值分解也可解释为每个股票在第二个因子中的因子暴露。在特征分解时按时间方向减去个股均值，但没有按个股方向减去市场均值。这时第一主成分每个元素的数值相近，相当于加权市场因子。因此第二主成分相当于与市场因子正交的第一个因子上个股的因子暴露。

因此 sector1 和 sector1 是相近的，都代表去掉市场因子后个股在第一个主要因子上的因子暴露排序，但由于两者市场因子的计算方式不同，结果不完全一致。

3.10 使用奇异值分解方法分析 3.4.4 节案例分析中利率曲线和期限结构数据（不进行按列去均值），并对比主成分分析结果。

答: 奇异值分解的第一主成分和均值特征接近但不完全一致，奇异值分解的第二主成分和特征分解的第二主成分接近但不完全一致。奇异值分解的第三主成分和特征分解的第三主成分接近但不完全一致。综上，奇异值分解忽略了特征分解的第一主成分的信息。

```
import numpy.linalg as la
## 特征分解
covx = np.cov(data.T)
u,v = la.eig(covx)
# 均值特征
```

```
mu = np.mean(data,axis=0)
plt.plot(mu)
# 第一主成分和主成分得分
plt.figure()
plt.plot(v[:,0])
# 第一主成分重构
xi = (data-mu).dot(v[:,0:1])
plt.figure()
rec_data = mu + xi.dot(v[:,0:1].T)
plt.plot(rec_data.T)
# 第二主成分和主成分得分
plt.figure()
plt.plot(v[:,1])
# 第二主成分重构
xi = (data-mu).dot(v[:,1:2])
plt.figure()
rec_data = mu + xi.dot(v[:,1:2].T)
plt.plot(rec_data.T)

## 奇异值分解
u2,d2,v2 = la.svd(data)
# 第一主成分
plt.figure()
plt.plot(-v2[0,:])
# 第二主成分
plt.figure()
plt.plot(-v2[1,:])
```

3.11 使用奇异值分解方法分析 3.4.5 节案例分析中股票收益率数据，并对比主成分分析结果。

答: 减去均值后使用奇异值分解方法与主成分分析的结果是完全一致的。

```
## 特征分解
covx = np.cov(retx.T)
u,v = la.eig(covx)
print(v)
print(v[:,1])

## 奇异值分解
u2,d2,v2 = la.svd(retx-np.mean(retx,axis=0))
```

```
print(v2)
print(v2[1,:])
```

3.12 理解并说明主成分分析、特征分解、奇异值分解之间的关系。

答: 见 3.4 节。

3.13 计算高斯混合模型中，未知参数的个数。

答: 一元高斯混合模型的公式为：

$$f(x;\theta)=\sum_{k=1}^{K}\pi_k f_k(x)=\sum_{k=1}^{K}\pi_k\phi(x;\mu_k,\sigma_k)$$

未知参数 $\theta=(\pi_1,...\pi_K,\mu_1,...,\mu_K,\sigma_1,...,\sigma_K)$，共 3K 个，并且有约束 $\sum_{k=1}^{K}\pi_k=1$。所以一元高斯混合模型中共有未知参数 3K-1 个。类似可计算多元高斯混合模型的参数。

3.14 在隐马尔科夫模型中，给出向后概率 β_{tj} 的计算下溢问题的处理方法。

答: 采用正则化方法，即 $\hat{\beta}_{tj}=c_t\beta_{tj}$。在 Baum-Welch 算法中，E 步结果中的 α_{tj} 和 β_{tj} 可以分别用 $\hat{\alpha}_{tj}$ 和 $\hat{\beta}_{tj}$ 替代，从而对所有参数估计完成重估计，注意到，重估计之后参数值并不会发生改变。对于似然函数 $logL_T=\sum_{t=1}^{T}logc_t$ 中的 $logL_T$，可以被用于验证每一步的迭代中似然函数值是否增加。

3.15 证明公式 (3.5.19)。

$$\hat{\alpha}_{tj}=c_1c_2\cdots c_t\alpha_{tj} \tag{3.5.19}$$

答:

$$\alpha_{tj}=\sum_{k=1}^{S}\alpha_{t-1,k}\gamma_k p_j(y_t|\theta_j)$$

令：

$$c_t=\frac{1}{\sum_{j=1}^{S}\tilde{\alpha}_{tj}}$$

对于 $j=1,...,S$，计算：

$$\hat{\alpha}_{tj}=c_t\tilde{\alpha}_{tj}$$

显然，$\hat{\alpha}_{1j}$=$c_1\alpha_{1j}$，假设：

$$\hat{\alpha}_{tj}=c_1c_2...c_t\alpha_{tj}$$

则：

$$\begin{aligned}\hat{\alpha}_{t+1,j} &= c_{t+1}\tilde{\alpha}_{t+1,j} \\ &= c_{t+1}\sum_{k=1}^{S}\hat{\alpha}_{t,k}\gamma_{kj}P(y_{t+1}|\theta_j) \\ &= c_1c_2...c_{t+1}\sum_{k=1}^{S}\alpha_{t,k}\gamma_{kj}P(y_{t+1}|\theta_j) \\ &= c_1c_2...c_{t+1}\alpha_{t+1,j}\end{aligned}$$

因此，通过归纳法可知对于所有 t，公式 (3.5.19) 成立。

4 线性回归和正则化方法

4.1 在回归分析流程的第 (1) 步中，如何根据剖面极大似然法来选择 Box-Cox 变换的参数 λ?

答: 以正态分布为例，假设转换后的 $\boldsymbol{y}$ 服从正态分布，$\boldsymbol{y}(\lambda) \sim \mathrm{N}\left(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_n\right)$。$\{(\mathbf{X}, \boldsymbol{y})\}$ 是观测到的数据，$\left(\lambda, \boldsymbol{\beta}, \sigma^2\right)$ 是模型参数。

$\boldsymbol{y}(\lambda)$ 的密度函数是

$$f(\boldsymbol{y}(\lambda)) = \frac{\exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{y}(\lambda) - \mathbf{X}\boldsymbol{\beta})'(\boldsymbol{y}(\lambda) - \mathbf{X}\boldsymbol{\beta})\right)}{(2\pi\sigma^2)^{\frac{n}{2}}}$$

令 $\boldsymbol{y}$ 到 $\boldsymbol{y}(\lambda)$ 的雅可比转换为 $J(\lambda, \boldsymbol{y})$，则 $\boldsymbol{y}$ 分密度函数 (模型的似然) 为

$$L\left(\lambda, \boldsymbol{\beta}, \sigma^2 \mid \boldsymbol{y}, \mathbf{X}\right) = f(\boldsymbol{y}) = \frac{\exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{y}(\lambda) - \mathbf{X}\boldsymbol{\beta})'(\boldsymbol{y}(\lambda) - \mathbf{X}\boldsymbol{\beta})\right)}{(2\pi\sigma^2)^{\frac{n}{2}}} J(\lambda, \boldsymbol{y})$$

为了得到最大似然估计，使用剖面极大似然：

(a) 给定 λ 求 $\tilde{\boldsymbol{\beta}}(\lambda)$ 和 $\hat{\sigma}^2(\lambda)$。此时极大化似然，可以得到

$$\begin{aligned}\tilde{\boldsymbol{\beta}}(\lambda) &= \left(\mathbf{X}'\mathbf{X}\right)^{-}\mathbf{X}\boldsymbol{y}(\lambda) \\ \hat{\sigma}^2(\lambda) &= \frac{\boldsymbol{y}(\lambda)'\left(\mathbf{I}_n - \mathbf{G}\right)\boldsymbol{y}(\lambda)}{n}\end{aligned}$$

其中 $\mathbf{G} = \mathbf{X}\left(\mathbf{X}'\mathbf{X}\right)^{-}\mathbf{X}'$。

(b) 将 $\tilde{\boldsymbol{\beta}}(\lambda)$ 和 $\hat{\sigma}^2(\lambda)$ 代入似然方程，得到剖面对数似然（即在给定 $\left(\boldsymbol{\beta}, \sigma^2\right)$ 情况下极大化似然函数 $l_P(\lambda)$）。注意，对于 Box-Cox 变换的原始形式，$J(\lambda, \boldsymbol{y}) = \prod_{i=1}^{n} y_i^{\lambda-1}$

$$\begin{aligned}l_P(\lambda) &= l\left(\lambda \mid \boldsymbol{y}, \mathbf{X}, \tilde{\boldsymbol{\beta}}(\lambda), \hat{\sigma}^2(\lambda)\right) \\ &= C - \frac{n}{2}\log\left(\hat{\sigma}^2(\lambda)\right) + (\lambda - 1)\sum_{i=1}^{n}\log\left(y_i\right)\end{aligned}$$

设 g 为 y 的几何平均 (i.e., $g = (\prod_{i=1}^{n} y_i)^{\frac{1}{n}}$), $\boldsymbol{y}(\lambda, g) = \frac{\boldsymbol{y}(\lambda)}{g^{\lambda-1}}$，则有

$$l_P(\lambda) = C - \frac{n}{2}\log\left(s_\lambda^2\right)$$

其中 s_λ^2 为模型 $\boldsymbol{y}(\lambda, g) \sim \mathrm{N}\left(\mathbf{X}\boldsymbol{\beta}, \sigma^2\mathbf{I}_n\right)$ 的残差平方和除以 n。所以最大化剖面似然, 我们只需要找到一个 λ 最小化 $s_\lambda^2 = \frac{\boldsymbol{y}(\lambda,g)'(\mathbf{I}_n - \mathbf{G})\boldsymbol{y}(\lambda,g)}{n}$。

4.2 在回归分析流程的第 (3) 步中, 如果使用中位数回归，后续步骤需要做哪些修正？

答: 第四步，不需要检验残差是否符合正态分布，不需要检验同方差，因为分位数回归中没有这些假设，相对结果更为稳定。

第五步，模型诊断，异常值可以以中位数的标准来找，如 S-H.ESD 方法。

第六步，模型推断，在估计 Y 置信区间的时候，可以考虑使用 bootstrap 方法。

4.3 在例 4.1 中，编写计算 R^2 的程序。

```
def R_square(X, y, beta_ols):
    yhat = X.dot(beta_ols)
    SSE = np.sum((yhat-y)**2)
    SST = np.sum((y-np.mean(y))**2)
    R2 = 1-SSE/SST
    return R2
```

4.4 在例 4.1 中，使用 statsmodels 中的 $OLSInfluence.cooks\_distance()$ 函数计算 Cook 距离，并自编函数对比计算结果。

```
#(1) OLSInfluence.cooks_distance()
import statsmodels.api as sm
import statsmodels.stats.outliers_influence as infl
model = sm.OLS(y,X)
results = model.fit()
results.summary()
cook1 = infl.OLSInfluence(results).cooks_distance[0]

#(2) our own function
def cook_own(x,y,beta):
    yhat = X.dot(beta)
    e2 = np.square(y.ravel()-yhat.ravel())
    n,p = x.shape
    s2 = np.sum(e2)/(n-p)
    H = x.dot(la.inv(x.T.dot(x))).dot(x.T)
    h = np.array([H[i][i] for i in range(n)])
    D = e2*h/((1-h)**2*p*s2)
    return D
cook2 = cook_own(X,y,beta_ols)
```

```
#(3) comparison
yhat = X.dot(beta_ols)
plt.figure()
plt.scatter(yhat,cook1,marker = 'x',color='red')
plt.scatter(yhat,cook2,marker = '+',color='blue')
plt.show()
```

4.5 参考回归分析流程，完成违约率案例分析。

答: 由于之前已经得到个人消费支出的对数（log（pce））和未偿还个人消费贷款（debt）的残差与违约率（rate）8 阶滞后项有相关关系，故接下来对得到人消费支出的对数（log（pce）），违约率（rate）和未偿还个人消费贷款（debt）进行建模。

(a) 估计模型，得到人消费支出的对数，违约率和未偿还个人消费贷款之间关系为

$$debt_i = -5933.41 + 745.72 * log(pce) + 19.78 rate_{i-8}$$

模型的方差估计为 173.87。

(b) 残差图可以看出残差和自变量没有明显关系，没有同方差。QQ 图可以看出误差分布具有正态性。

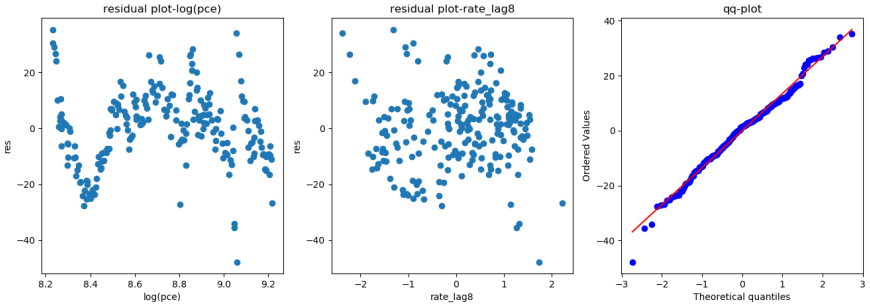

图 1: 习题 4.5 回归的残差图和 qq 图

(c) 模型诊断，可以看出杠杆值和库克距离最大分别为 0.046，0.108。

(d) 模型推断，例如置信区间和假设检验。模型的 $R^2 = 0.996$。

$\beta_{log(pce)}$ 的 p 值为 0，95% 置信区间为 [739.6339251155719, 751.8058012050842].

$\beta_{rate_{i-8}}$ 的 p 值为 0，95% 置信区间为 [17.95249970342674, 21.612364528916515].

代码如下：

```
import scipy.stats as st
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import numpy.linalg as la

# 读数据
data = pd.read_excel('macro_econ_data.xls')
debt = data['Revolving Credit'].values[:,np.newaxis]
pce = data['Nominal PCE'].values[:,np.newaxis]
x = np.log(pce)
y = debt

# 画出散点图观察趋势。
plt.figure()
plt.title('log(pce)-debt scatter plot')
plt.xlabel('log(pce)')
plt.ylabel('debt')
plt.plot(x,y,'o')
plt.savefig("scatter.png")

# 建立模型估计所需参数。
n,p = x.shape
c = np.ones((n,1))
X = np.hstack((c,x))
rate_lag8 = rate2[lag:]
X = np.hstack((X[:-lag],rate_lag8))
y = y[:-lag]。
beta = la.inv(X.T.dot(X)).dot(X.T).dot(y)
res = y - X.dot(beta)
sigmahat = np.sum(res**2)/(n - p + 1)
print('beta_hat = '+str(beta.T))
print('sigma2_hat = '+str(sigmahat))

# 观察残差图和qq 图。
#残差图
plt.figure()
plt.plot(x[:-lag], res,'o')
plt.xlabel('log(pce)')
```

```
plt.ylabel('res')
plt.title('residual plot')
plt.savefig("残差图-1.png")
plt.figure()
plt.plot(rate_lag8, res,'o')
plt.xlabel('rate_lag8')
plt.ylabel('res')
plt.title('residual plot')
plt.savefig("残差图-2.png")
#qq 图
from scipy import stats
plt.figure()
stats.probplot(res.ravel(),dist = "norm",plot = plt)
plt.title('qq-plot')
plt.savefig("qq图.png")

# 考虑是否存在杠杆点与强影响点。
#杠杆
H = X.dot(la.inv(X.T.dot(X))).dot(X.T)
lev = np.diag(H)
np.argsort(lev)[-5:]
print('最大的五个杠杆值: '+str(np.sort(lev)[-5:]))
#强影响点
def cook_own(x,y,beta_ols):
    yhat = x.dot(beta_ols)
    e2 = np.square(y.ravel()-yhat.ravel())
    n,p = x.shape
    s2 = np.sum(e2)/(n-p)
    H = x.dot(la.inv(x.T.dot(x))).dot(x.T)
    h = np.array([H[i][i] for i in range(n)])
    D = e2*h/((1-h)**2*p*s2)
return D
cooks = cook_own(X,y,beta)[5:,]
print('最大的五个cook距离: '+str(np.sort(cooks)[-5:]))

# 模型推断: 假设检验, 区间估计。
def R_square(X, y, beta_ols):
    yhat = X.dot(beta_ols)
    SSE = np.sum((yhat - y)**2)
    SST = np.sum((y - np.mean(y))**2)
    R2 = 1 - SSE/SST
```

```
return R2
print('R2 = '+str(R_square(X, y, beta)))
#置信区间
var_beta = sigmahat*la.inv(X.T.dot(X))
print('beta1的95%置信区间为' + str([(beta[1] -
    1.96*(var_beta[1,1]**0.5))[0],(beta[1] +
    1.96*(var_beta[1,1]**0.5))[0]]))
print('beta2的95%置信区间为' + str([(beta[2] -
    1.96*(var_beta[2,2]**0.5))[0],(beta[2] +
    1.96*(var_beta[2,2]**0.5))[0]]))
#t 检验
tstat1 = beta[1]/(var_beta[1,1]**0.5)
tstat2 = beta[2]/(var_beta[2,2]**0.5)
#p 值
p_value1 = st.t.cdf( - tstat1,n-p+1)*2
p_value2 = st.t.cdf( - tstat2,n-p+1)*2
print('beta1的p值为' + str(p_value1))
print('beta2的p值为' + str(p_value2))
```

4.6 编写向后选择法的程序并进行模拟仿真。

```
def backward0(X,y):
    n,p = X.shapeA
    seq = list(range(p))
    del_seq = []
    for j in range(p):
        minrss = float('inf')
        for i in range(len(seq)):
        temp_seq = seq[:]
        temp_i = temp_seq.pop(i)
        beta = la.pinv(X[:,temp_seq].T.dot(X[:,temp_seq]))
                .dot(X[:,temp_seq].T).dot(y)
        Z = y - X[:,temp_seq].dot(beta)
        rss = sum(Z**2)
        if rss < minrss:
            minrss = rss
            mini = temp_i
        seq.remove(mini)
        del_seq.append(mini)
    return del_seq
```

```
n = 2000
p = 10
x = np.random.rand(n,p)
beta_true = np.array(range(p))
error = np.random.randn(n,1)*0.3
y = x.dot(beta_true.reshape(p,1)) + error
del_seq = backward0(x,y)
print(del_seq)
```

4.7 在 4.2.2 节案例分析（指数跟踪）中，使用移动窗口的方法来回测跟踪误差。

```
# 数据生成
X = retx
y = np.mean(X,axis = 1).reshape(1339,1)
seq =  forward0(X[0:500,:],y[0:500,:])
id = seq[:50]
# 滚动窗口
pred=[]
for i in range(200):
    X2 = X[i:500+i,:]
    if i==0:
        y2 = y[i:500,:]
    else:
        y2 = np.append(y[i:500,:],pred[-i:]).reshape(500,1)
    beta =
        la.pinv(X2[:,id].T.dot(X2[:,id])).dot(X2[:,id].T).dot(y2)
    beta = beta/np.sum(beta)
    X3 = X[500+i,:]
    ret_test = X3[id].dot(beta)
    pred.append(ret_test)
pred = np.array(pred).reshape(200,1)
y3 = y[500:700,:]
plt.figure()
plt.plot(np.cumprod(1+y3),label='index return')
plt.plot(np.cumprod(1+pred),label='rolling portfolio return')
plt.legend()
```

4.8 尝试使用其它方法（如 Lasso、Forward Stagewise 等）来处理指数跟踪问题。

```
        X = retx # 来自案例 2.5.3
        y = np.mean(X,axis = 1).reshape(1339,1)
```

```
X2 = X[0:500,:]
y2 = y[0:500,:]
X3 = X[500:700,:]
y3 = y[500:700,:]

##FS
id = np.sum(X2,0)!=0
XX2 = X2[:,id]
# 拟合
beta =  fs(XX2,y2,0.01)[0]#见例4.3
# 验证跟踪效果
ret_test = X3[:,id].dot(beta)
plt.plot(np.cumprod(1+y3))
plt.plot(np.cumprod(1+ret_test))

##lasso
# 拟合
reg_lasso = linear_model.Lasso(alpha = 0.00001)
reg_lasso.fit(X2,y2)
# 验证跟踪效果
ret_test = reg_lasso.predict(X3[:,])
plt.plot(np.cumprod(1+y3))
plt.plot(np.cumprod(1+ret_test))
```

4.9 证明式 (4.3.2) 中的 λ 和式 (4.3.1) 中的 t 存在对应的关系。

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}x_{ij}\beta_j\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2 \tag{4.3.2}$$

$$\min_{\beta}\sum_{i=1}^{N}\left(y_i-\beta_0-\sum_{j=1}^{p}x_{ij}\beta_j\right)^2 \quad \text{s.t.} \quad \sum_{j=1}^{p}\beta_j^2\le t \tag{4.3.1}$$

答:

$$\min \sum_{i=1}^{n} \left(y_i - x_i^\top \beta\right)^2 - \lambda \left(\sum_{j=1}^{n} \beta_j^2 - t\right) \doteq \mathrm{Ln}(\beta;\lambda)$$

$$\Rightarrow \begin{cases} 0 = \frac{\partial \mathrm{Ln}(\beta_i,\lambda)}{\partial \beta} = 2\sum_{i=1}^{n}\left(y_i - x_i^\top \beta\right) x_i - 2\lambda\beta \\ 0 = \frac{\partial \mathrm{Ln}(\beta_i,\lambda)}{\partial \lambda} = \sum_{j=1}^{n} \beta_j^2 - t \end{cases}$$

$$\Rightarrow \begin{cases} \lambda\beta = x^\top y - x^\top x\beta \\ t = \beta^\top \beta \end{cases}$$

$$\Rightarrow \begin{cases} \beta = \left(x^\top x + \lambda I\right)^{-1} x^\top y \\ t = y^\top x \left(x^\top x + \lambda I\right)^{-1} \left(x^\top x + \lambda I\right) x^\top y \end{cases}$$

即 λ 和 t 的关系为 $t = y^\top x \left(x^\top x + \lambda I\right)^{-1} \left(x^\top x + \lambda I\right) x^\top y$

4.10 使用二次规划算法求解 Lasso 问题，并学习 CVXPY 模块。

```
import cvxpy as cp
def LP_lasso(X,y,t=100):
    n,p = X.shape
    # Define and solve the CVXPY problem.
    beta = cp.Variable(2*p)
    XX = np.hstack((X, np.zeros((n,p))))
    objective = cp.Minimize(cp.sum_squares(y.reshape((n,)) - XX @
        beta))
    constraints = [cp.sum(beta[p:]) <= t,
    beta[p:] >= 0,
    -beta[p:] <= beta[:p], beta[:p] <= beta[p:]] #element-wise
    prob = cp.Problem(objective, constraints)
    prob.solve()
    return beta.value[:p].copy(),prob.value

# Generate data
n = 100
beta_true = np.array([1,-2,3,-4,5,-6,7,-8])
p = len(beta_true)
X = np.random.randn(n,p)
error = np.random.randn(n,1)*0.3
y = X.dot(beta_true.reshape(p,1)) + error
beta,prob = LP_lasso(X,y,t=100)
print("\nThe optimal value is", prob)
print("A solution beta is")
```

```
print(beta)
```

4.11 编写 Lasso 问题的局部二次近似算法的程序，并进行模拟仿真。

```
def local_quadratic_appro(X,y,beta0,lam):
    old_beta = beta0.copy()
    n,p = X.shape
    iter = 0
    diff = 1
    VAL = 10000
    while iter<1000 and diff>0.0001:
        diag = np.diag(lam / np.squeeze(np.abs(old_beta)))
        beta = la.pinv(X.T.dot(X) + n * diag).dot(X.T).dot(y)
        VAL2 = np.sum((y-X.dot(beta))**2) +
            n*beta.T.dot(diag).dot(beta)
        diff = np.abs(VAL2 - VAL)
        VAL = VAL2
        old_beta = beta
        iter = iter + 1
    return beta,iter,diff

# Generate data
n = 100
beta_true = np.array([1,-2,3,-4,5,-6,7,-8])
p = len(beta_true)
X = np.random.randn(n,p)
error = np.random.randn(n,1)*0.3
y = X.dot(beta_true.reshape(p,1)) + error
beta_ols = la.pinv(X.T.dot(X)).dot(X.T).dot(y) #作为迭代的初值
lam = 0.01
beta_appro, actual_iter, diff = local_quadratic_appro(X, y,
    beta_ols, lam)
print("\nThe iteration is",actual_iter)
print("A solution beta is")
print(beta_appro)
```

4.12 通过模拟仿真对比本章提到的 Lasso 问题的 4 个算法的效率。

答: 模拟重复 100 次，每次模拟都随机生成同样的数据（见代码），分别用四种方法计算 MSE = 运行时间 $Time$，100 次的平均值和最大值结果如表 2。从前两行 MSE 可以看出，四种算法计算精度差不多，后两行 $Time$ 可以看出，从计算时间来看，坐标

下降优于 LARS 优于局部二次近似优于线性规划。

表 2: Lasso 问题 4 个算法的效率对比

	坐标下降	局部二次近似	线性规划	LARS
$MSE_{mean}(10^{-3})$	7.72316	7.72288	7.5701	7.69901
$MSE_{mean}(10^{-3})$	17.0898	17.0944	17.5859	18.1657
$Time_{mean}(s)$	0.055	0.325	1.25	0.081
$Time_{max}(s)$	0.097	0.569	1.442	0.108

```
from sklearn.model_selection import RepeatedKFold
from sklearn.linear_model import LarsCV
from sklearn import linear_model
from sklearn.model_selection import KFold

#CD 直接调包
#LQA,LP 的函数见习题4.10 和4.11 答案
# LARS
def LARS(X,y):
    # define model evaluation method
    cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
    # define model
    reg = LarsCV(cv=cv, n_jobs=-1)
    # fit model
    reg.fit(X, y)
    beta = reg.coef_
return beta

# 总函数
def regression(X,y,lamb,method):
    beta_ols = la.pinv(X.T.dot(X)).dot(X.T).dot(y)
    if method == 'CD':
        reg = linear_model.Lasso(alpha = lamb,fit_intercept=False)
        reg.fit (X, y)
        beta_hat = reg.coef_
    elif method == 'LQA':
        beta_hat = local_quadratic_appro(X,y, beta_ols, lamb)[0]
    elif method == 'LP':
        bb =
            y.T.dot(X).dot(la.pinv(X.T.dot(X)+lamb*np.eye(X.shape[1])))
```

```
        t = bb.dot(bb.T)
        beta_hat = LP_lasso(X,y,t)[0]
    else:
        beta_hat = LARS(X,y)
    return beta_hat

# CV
def CV(X,y,method,lamb_list,kfold = 5):
    kf = KFold(kfold, True, 10)
    cv_arr = []
    for lamb in lamb_list:
        cv =  0
        for train_index, test_index in kf.split(X):
            X_train = X[train_index]
            Y_train = y[train_index]
            X_test = X[test_index]
            Y_test = y[test_index]
            beta_hat = regression(X_train,Y_train,lamb,method)
            Y_pre = X_test.dot(beta_hat)
            loss  =
                np.mean((Y_test.reshape(-1)-Y_pre.reshape(-1))**2)
            cv = cv + loss
        cv_arr.append(cv)
    index = np.argmin(np.array(cv_arr))
    lamb_best  = lamb_list[index]
    return lamb_best

# simulation
def generate_data(n,beta_true):
    p = len(beta_true)
    X = np.random.randn(n,p)
    error = np.random.randn(n,1)*0.3
    y = X.dot(beta_true.reshape(p,1)) + error
    return X,y

import time
import random
n = 100
ntimes = 100
beta_true = np.array([1,-2,3,-4,5,-6,7,-8])
lamb_list = np.exp(np.linspace(-6,1,20))
```

```
beta_all = np.zeros((p,4,ntimes))
lds = np.zeros((4,ntimes))
times = np.zeros((4,ntimes))
for i in range(ntimes):
    random.seed(i)
    np.random.seed(i)
    X,y = generate_data(n,beta_true)
    meths = ['CD','LQA','LP','LARS']
    for j in range(4):
        print(i,j)
        method = meths[j]
        start = time.time()
        if j == 3:
            lamb = 0
        else:
            lamb = CV(X,y,method,kfold = 5)
        beta_all[:,j,i] = regression(X,y,lamb,method).ravel()
        end = time.time()
        times[j,i] = end - start
        lds[j,i] = lamb

print('四种方法的mse: ',
        np.mean(np.sum((beta_all-beta_true.reshape(8,1,1))**2,0),1))
print('四种方法的运行时长',np.mean(times,1))
print('四种方法的最大mse: ',
        np.max(np.sum((beta_all-beta_true.reshape(8,1,1))**2,0),1))
print('四种方法的最长运行时长',np.max(times,1))
```

4.13 弹性网 L1+L2 正则作为惩罚函数是否具有稀疏性？

答: 弹性网具有稀疏性。

弹性网惩罚函数为 $P_\lambda(\beta) = \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2$，求导有 $P'_\lambda(|\beta|) = \lambda_1 + \lambda_2|\beta|$.

取 $c_0 = \lambda_1$ 即可满足稀疏性充分条件 $\min_\beta \{|\beta| + P'_\lambda(|\beta|)\} = c_0 > 0$.

4.14 对比分析岭回归和主成分回归中奇异值分解的作用。

答: 奇异值分解在岭回归中对每个主成分压缩，在主成分回归中选择特征值最大的前 q 个主成分。$\hat{\beta}_j^{ols} = \sum_{k=1}^{p} v_{jk}\hat{\alpha}_k/d_k$ 则

(a) 岭回归中

$$\hat{\beta}_j^{ridge} = \sum_{k=1}^{p} v_{jk} d_k \hat{\alpha}_k / (d_k^2 + \lambda)$$

(b) 主成分回归中

$$\hat{\beta}_j^{pcr} = \sum_{k=1}^{q} v_{jk} \hat{\alpha}_k / d_k$$

4.15 修改 QR 算法的程序，使得特征值为正数且有序。

答: 可直接对原 QR 算法程序的输出进行进一步处理，代码如下

```
def sort_eig_qr(u,v):
p = len(u)
u2 = np.zeros((p,))
v2 = np.zeros((p,p))
u_abs = np.abs(u)
id = np.argsort(u_abs)
for j in range(p):
if u[id[j]]>0:
u2[j] = u[id[j]]
v2[:,j] = v[:,id[j]]
if u[id[j]]<0:
u2[j] = -u[id[j]]
v2[:,j] = -v[:,id[j]]
return u2,v2
```

5 分类

5.1 对 LDA 模拟仿真生成的数据使用 QDA 方法进行分析，画出决策边界，并与 LDA 的结果进行比较。

答: 见 5.2。

5.2 对 LDA 模拟仿真生成的数据使用朴素贝叶斯方法进行分析，画出决策边界，并与 LDA 和 QDA 的结果进行比较。

答: 比较结果如图 2，可以看出

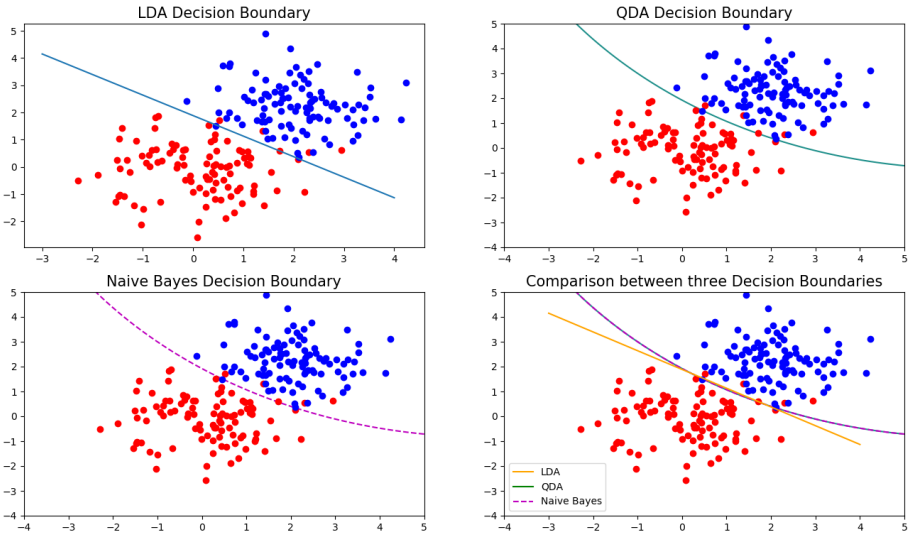

图 2: 习题 5.2 LDA 和 QDA 以及朴素贝叶斯方法对比

(a) LDA 产生的决策边界是直线/平面，QDA 与朴素贝叶斯产生的决策边界是曲线/曲面。

(b) 由于数据生成模型的设计，实验所用的两类数据拥有不同的均值以及相同的协方差矩阵，且每一类多元样本的分布中的每个特征是独立的，所以这三类分类方法在该数据集上表现相近，决策边界也十分相似。

```
""" LDA """
```

```
## LDA estimation
p1 = 0.5
p2 = 0.5
mu1 = np.mean(x1,axis = 0)
mu2 = np.mean(x2,axis = 0)
S = (np.cov(x1.T)*99 + np.cov(x2.T)*99)/198
## LDA Classification and Boundary
delta1 = X.dot(la.inv(S)).dot(mu1) -
    0.5*mu1.dot(la.inv(S)).dot(mu1)
delta2 = X.dot(la.inv(S)).dot(mu2) -
    0.5*mu2.dot(la.inv(S)).dot(mu2)
id  = delta1 > delta2
b0 = 0.5*mu1.dot(la.inv(S)).dot(mu1) -
    0.5*mu2.dot(la.inv(S)).dot(mu2)
b = (la.inv(S)).dot(mu1-mu2)
u = np.linspace(-3,4,100)
fu = b0/b[1] - b[0]/b[1]*u

""" QDA """
## QDA estimation
p1 = 0.5
p2 = 0.5
mu1 = np.mean(x1,axis = 0)
mu2 = np.mean(x2,axis = 0)
S1 = np.cov(x1.T)
S2 = np.cov(x2.T)
## QDA Classification and Boundary
delta1_q = np.zeros((200,))
delta2_q = np.zeros((200,))
for i in range(X.shape[0]):
    delta1_q[i] = -0.5*np.log(la.det(S1))-0.5*(X[i,
        :]-mu1).dot(la.inv(S1)).dot((X[i, :]-mu1).T)
    delta2_q[i] = -0.5*np.log(la.det(S2))-0.5*(X[i,
        :]-mu2).dot(la.inv(S2)).dot((X[i, :]-mu2).T)
idQ  = delta1_q > delta2_q
invS1 = la.inv(S1)
invS2 = la.inv(S2)
b0 = -np.log(la.det(S1)) + np.log(la.det(S2))
xlist = np.linspace(-4.0, 5.0, 1000)
ylist = np.linspace(-4.0, 5.0, 1000)
plotX,plotY = np.meshgrid(xlist, ylist)
```

```
Z = b0 - (invS1[0][0]*(plotX-mu1[0])**2 +
    (invS1[0][1]+invS1[1][0])*(plotX-mu1[0])*(plotY-mu1[1]) +
    invS1[1][1]*(plotY-mu1[1])**2) +
    (invS2[0][0]*(plotX-mu2[0])**2 +
    (invS2[0][1]+invS2[1][0])*(plotX-mu2[0])*(plotY-mu2[1]) +
    invS2[1][1]*(plotY-mu2[1])**2)

""" 朴素贝叶斯 """
## Naive Bayes estimation
mu11 = np.mean(x1[:,0])
mu12 = np.mean(x1[:,1])
mu21 = np.mean(x2[:,0])
mu22 = np.mean(x2[:,1])
S11 = np.var(x1[:,0], ddof = 1)
S12 = np.var(x1[:,1], ddof = 1)
S21 = np.var(x2[:,0], ddof = 1)
S22 = np.var(x2[:,1], ddof = 1)
## Naive Bayes Classification and Boundary
delta1_nb = - np.log(S11) - np.log(S12) -
    ((X[:,0]-mu11)**2)/(2*S11) - ((X[:,1]-mu12)**2)/(2*S12)
delta2_nb = - np.log(S21) - np.log(S22) -
    ((X[:,0]-mu21)**2)/(2*S21) - ((X[:,1]-mu22)**2)/(2*S22)
idnb = delta1_nb > delta2_nb
b0 = - np.log(S11) - np.log(S12) + np.log(S21) + np.log(S22)
xlist = np.linspace(-4.0, 5.0, 1000) # Create 1-D arrays for x,y
    dimensions
ylist = np.linspace(-4.0, 5.0, 1000)
pX,pY = np.meshgrid(xlist, ylist)
Z = b0 - ((pX-mu11)**2)/(2*S11) - ((pY-mu12)**2)/(2*S12) +
    ((pX-mu21)**2)/(2*S21) + ((pY-mu22)**2)/(2*S22)

#作图比较
plt.figure(figsize = (16,9))
plt.subplot(2,2,1)
plt.plot(x1[:,0],x1[:,1],'ro')
plt.plot(x2[:,0],x2[:,1],'bo')
plt.plot(u,fu)
plt.title('LDA Decision Boundary',fontsize = 15)
plt.subplot(2,2,2)
plt.plot(x1[:,0],x1[:,1],'ro')
plt.plot(x2[:,0],x2[:,1],'bo')
```

```
plt.contour(plotX, plotY, Z, 0, linestyles = 'solid')
plt.title('QDA Decision Boundary',fontsize = 15)
plt.subplot(2,2,3)
plt.plot(x1[:,0],x1[:,1],'ro')
plt.plot(x2[:,0],x2[:,1],'bo')
plt.contour(pX, pY, Z, 0, colors = 'm', linestyles = '--')
plt.title('Naive Bayes Decision Boundary',fontsize = 15)
plt.subplot(2,2,4)
plt.plot(x1[:,0],x1[:,1],'ro')
plt.plot(x2[:,0],x2[:,1],'bo')
plt.plot(u,fu,'orange',label='LDA')
plt.plot(-3,-2,'green',ls = '-',label='QDA')
plt.plot(-3,-2,'m',ls = '--',label='Naive Bayes')
plt.contour(plotX, plotY, Z, 0, linestyles = 'solid')
plt.contour(pX, pY, Z, 0, colors = 'm', linestyles = '--')
plt.legend(loc='lower left')
plt.title('Comparison between three Decision Boundaries',fontsize
    = 15)
plt.show()
plt.savefig("cla-comparision.png")
```

5.3 写出带 L2 惩罚的逻辑回归的 Python 程序。

答: 见 5.4。

5.4 对玩具数据 $(X, Y) : \{(-3, 0), (-2, 0), (-1, 0), (1, 1), (2, 1), (3, 1)\}$，使用带有 L2 惩罚项和不带有 L2 惩罚项的逻辑回归来估计参数，并观察结果。

答: 运行玩具数据可以看出，不带惩罚的逻辑回归得到的 β 一直增大至无穷带 L2 惩罚的逻辑回归则较快收敛，模拟结果与前文所述结论一致，“即当数据完全线性可分时，如果没有对参数的惩罚，估计参数值会趋于无穷大，解决方案是对似然函数加上一个 L2 惩罚项；所以在很多软件对逻辑回归的求解中 l2 正则化是自带的（例如 sklearn）。”

```
def logistic2(X,y,beta,lam = 0,penalty = 'l2',k = 0.5, iter_upper
    = 10000,intercept = True):
    diff,diffb = 1,1
    iter = 0
    lenb = beta.shape[0]
    beta2 = beta.copy()
    while iter <iter_upper and (diff >1e-8 or diffb >1e-8):
        beta = beta2
```

```
        like = np.sum((y*X.dot(beta)) -
            np.log(1+np.exp(X.dot(beta))))
        p = np.exp(X.dot(beta))/(1+np.exp(X.dot(beta)))
        w = np.diag((p*(1-p)).ravel())
        l1_1 = np.sign(beta)  #l1的一阶导
        l1_2 = 0#l1的二阶导
        l2_1 = 2*beta.copy()#l2的一阶导
        l2_2 = 2*np.eye(lenb)#l2的二阶导
        if penalty=='l2':
            k = 1
        elif penalty=='l1':
            k = 0
        elif penalty=='enet':
            k = 0.5
        mat1 = k*l2_1 +(1-k)*l1_1
        mat2 = k*l2_2 +(1-k)*l1_2
        if intercept:
            #有截距项，则不对截距项惩罚，即截距项的导数设置为0
            mat1[0][0] = 0
            mat2[0] = 0
        beta2 = beta +
            la.pinv(X.T.dot(w).dot(X)+lam*mat2).dot((X.T).dot(y-p)-lam*mat1)
        like2 = np.sum((y*X.dot(beta2)) -
            np.log(1+np.exp(X.dot(beta2))))
        diff = np.abs(like - like2)
        diffb = np.sum(beta - beta2)**2
        iter = iter + 1
        print(iter,diff,diffb,beta2)
    return beta

#toy data
toy = np.array([[-3,0],[-2,0],[-1,0],[1,1],[2,1],[3,1]])
toy_X = toy[:,0].reshape(6,1)
toy_y = toy[:,1].reshape(6,1)
beta0_toy = la.inv(toy_X.T.dot(toy_X)).dot(toy_X.T).dot(toy_y)
beta_toy = logistic2(toy_X,toy_y,beta0_toy,lam = 0,iter_upper =
    1000, intercept = False)
print('不带L2约束的参数估计结果')
print(beta_toy)
beta_toy_l2 = logistic2(toy_X,toy_y,beta0_toy, penalty = 'l2',
    lam=0.1, intercept = False)
```

```
print('带L2约束的参数估计结果')
print(beta_toy_l2)
beta_toy_l2 = logistic2(toy_X,toy_y,beta0_toy, penalty = 'l1',
    lam=0.1, intercept = False)
print('带L1约束的参数估计结果')
print(beta_toy_l2)
beta_toy_l2 = logistic2(toy_X,toy_y,beta0_toy, penalty = 'enet',
    lam=0.1, intercept = False)
print('带L1+l2约束的参数估计结果')
print(beta_toy_l2)
```

5.5 写出带 L1+L2 惩罚的逻辑回归的 Python 程序。

答: 见 5.4。

5.6 对股票涨跌数据，分别使用 LDA，QDA 和朴素贝叶斯方法进行分析，并与逻辑回归的结果进行比较。

答: LDA,QDA,NB 的 IC 分别为 $-0.0535, 0.1321, -0.0517$ 与逻辑回归的 0.077 相比，只有 QDA 的分类结果优于逻辑回归，而 LDA 和 NB 的分类效果并不好。

```
""" LDA """
## LDA estimation
def LDA(X,y,Xpre):
    x1 = X[y==1,:]
    x2 = X[y==0,:]
    mu1 = np.mean(x1,axis = 0)
    mu2 = np.mean(x2,axis = 0)
    S = (np.cov(x1.T)*99 + np.cov(x2.T)*99)/198
    delta1 = Xpre.dot(la.inv(S)).dot(mu1) -
        0.5*mu1.dot(la.inv(S)).dot(mu1)
    delta2 = Xpre.dot(la.inv(S)).dot(mu2) -
        0.5*mu2.dot(la.inv(S)).dot(mu2)
    ypre  = delta1 > delta2
    return ypre
X,y,Xpre = X_train,y_train,X_test
y_pred = LDA(X_train,y_train,X_test)
np.corrcoef([y_test,y_pred])
print('IC if LDA is',np.corrcoef([y_test,y_pred])[0,1])
```

```
""" QDA """
## QDA estimation
def QDA(X,y,Xpred):
    x1 = X[y==1,:]
    x2 = X[y==0,:]
    mu1 = np.mean(x1,axis = 0)
    mu2 = np.mean(x2,axis = 0)
    S1 = np.cov(x1.T)
    S2 = np.cov(x2.T)
    delta1_q = np.zeros((Xpred.shape[0],))
    delta2_q = np.zeros((Xpred.shape[0],))
    for i in range(Xpred.shape[0]):
        delta1_q[i] = -0.5*np.log(la.det(S1))-0.5*(Xpred[i,
            :]-mu1).dot(la.inv(S1)).dot((Xpred[i, :]-mu1).T)
        delta2_q[i] = -0.5*np.log(la.det(S2))-0.5*(Xpred[i,
            :]-mu2).dot(la.inv(S2)).dot((Xpred[i, :]-mu2).T)
    idQ  = delta1_q > delta2_q
    return idQ
X,y,Xpre = X_train,y_train,X_test
y_pred = QDA(X_train,y_train,X_test)
np.corrcoef([y_test,y_pred])
print('IC if QDA is',np.corrcoef([y_test,y_pred])[0,1])

""" 朴素贝叶斯 """
## Naive Bayes estimation
def NB(X,y,Xpred):
    x1 = X[y==1,:]
    x2 = X[y==0,:]
    mu11 = np.mean(x1[:,0])
    mu12 = np.mean(x1[:,1])
    mu21 = np.mean(x2[:,0])
    mu22 = np.mean(x2[:,1])
    S11 = np.var(x1[:,0], ddof = 1)
    S12 = np.var(x1[:,1], ddof = 1)
    S21 = np.var(x2[:,0], ddof = 1)
    S22 = np.var(x2[:,1], ddof = 1)
    ## Naive Bayes Classification and Boundary
    delta1_nb = - np.log(S11) - np.log(S12) -
        ((Xpred[:,0]-mu11)**2)/(2*S11) -
        ((Xpred[:,1]-mu12)**2)/(2*S12)
    delta2_nb = - np.log(S21) - np.log(S22) -
```

```
        ((Xpred[:,0]-mu21)**2)/(2*S21) -
        ((Xpred[:,1]-mu22)**2)/(2*S22)
    idnb = delta1_nb > delta2_nb
    return idnb
y_pred = NB(X_train,y_train,X_test)
np.corrcoef([y_test,y_pred])
print('IC if NB is',np.corrcoef([y_test,y_pred])[0,1])
```

5.7 在支持向量机中，证明 $(\mathbf{x}_i^T\beta+\beta_0)/\parallel\beta\parallel$ 是样本点 $\mathbf{x}_i$ 到超平面 S 的带符号距离。（提示：参考《ESL》4.5 节）

答: (a) 超平面 S 上任意两点 $\mathbf{x}_1$ 和 $\mathbf{x}_2$，有 $\beta^T(\mathbf{x}_1-\mathbf{x}_2)=0$, 且 $\beta^*=\beta/\|\beta\|$ 是与超平面 S 正交的一个单位向量。

(b) 任意超平面 S 上一点 $\mathbf{x}_0$ 有 $\beta^T\mathbf{x}_0=-\beta_0$.

(c) 故样本点 $\mathbf{x}_i$ 到超平面 S 的带符号距离为

$$\beta^{*T}(\mathbf{x}_i-\mathbf{x}_0)=\frac{1}{\|\beta\|}\left(\beta^T\mathbf{x}_i+\beta_0\right).$$

5.8 使用支持向量机分别分析 LDA 模拟仿真生成的数据和股票涨跌数据。

答: 仿真数据分类结果如图 3。股票涨跌数据中 IC 为 0.811，大于逻辑回归的 0.077。

```
#模拟仿真数据
from sklearn import svm
import matplotlib
clf = svm.LinearSVC()
clf.fit(X,y.ravel())
y_hat = clf.predict(X) == 1
x1_min, x1_max = X[:, 0].min(), X[:, 0].max()  # 第0列的范围
x2_min, x2_max = X[:, 1].min(), X[:, 1].max()  # 第1列的范围
t1 = np.linspace(x1_min, x1_max, 2000)
t2 = np.linspace(x2_min, x2_max, 2000)
plt.figure()
xx1, xx2 = np.meshgrid(t1, t2)
grid_test = np.stack((xx1.flat, xx2.flat), axis=1)
grid_hat = clf.predict(grid_test)
plt.title('SVM Classification Boundary',fontsize = 15)
plt.plot(x1[:,0],x1[:,1],'ro')
plt.plot(x2[:,0],x2[:,1],'bo')
```

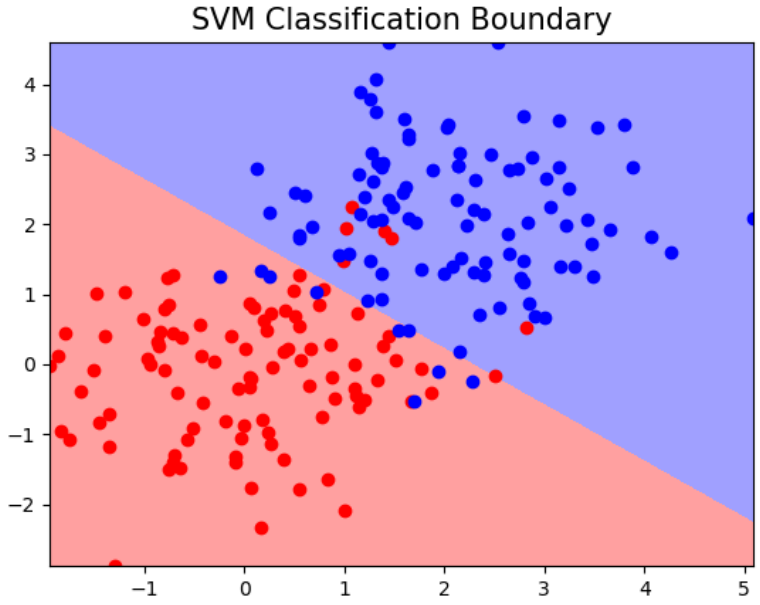

图 3: 习题 5.8 LDA 和 QDA 以及朴素贝叶斯方法对比

```
cm_light = matplotlib.colors.ListedColormap([ '#A0A0FF',
    '#FFA0A0'])
plt.pcolormesh(xx1, xx2, grid_hat.reshape(xx1.shape), cmap =
    cm_light)
plt.show()
plt.savefig("cla-svm.png")

#股票涨跌数据
## 拟合模型
from sklearn import svm
from sklearn.metrics import classification_report
clf = svm.LinearSVC()
clf.fit(X_train,y_train)
## 训练样本的表现
y_pred0 = clf.predict(X_train)
print(classification_report(y_train, y_pred0))
np.corrcoef([y_train,y_pred0])
## 检验样本的表现
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
# Information Coefficient, IC
np.corrcoef([y_test,y_pred])
print('IC if SVM is',np.corrcoef([y_test,y_pred])[0,1])
```

5.9 自主编程计算股票涨跌数据的各种评判度量，如 F1 得分，ROC 和 AUC 等。

```
class our_metrics:
    def __init__(self, y, y_pred, y_pred_prob = None):
        self.y = y
        self.y_pred = y_pred
        self.y_pred_prob = y_pred_prob

    def confusion_matrix(self):
        S = len(self.y)
        PO = np.sum(self.y)
        NO = S - PO
        PX = np.sum(self.y_pred)
        NX = S - PX
        TP = np.sum(self.y_pred*self.y)
        FP = PX - TP
        FN = PO - TP
        TN = NO - FP  # = NX - FN
        # Confusion Matrix
        self.confusion_matrix =
            np.array([[TP,FP],[FN,TN]]).astype(int)
        ## F1 Score
        TPR = TP/(TP + FN)
        PPV = TP/(TP + FP)
        self.TPR = TPR
        self.PPV = PPV
        self.F1 = 2*TPR*PPV/(TPR + PPV)

    def ROC(self):
        fpr_seq = []
        tpr_seq = []
        S = len(self.y)
        PO = np.sum(self.y)
        NO = S - PO
        for ld in np.linspace(0.0,0.99,5000):
            y_temp = (self.y_pred_prob>ld).astype(int)
            PX = np.sum(y_temp)
```

```
            NX = S - PX
            TP = np.sum(y_temp*self.y)
            FP = PX - TP
            FN = PO - TP
            TN =  NX - FN
            TPR = TP/(TP + FN)
            FPR = FP/(TN + FP)
            fpr_seq.append(FPR.copy())
            tpr_seq.append(TPR.copy())
        fpr_seq2 = np.asarray(fpr_seq)[::-1]
        tpr_seq2 = np.asarray(tpr_seq)[::-1]

        fpr_seq3 = np.hstack((0,fpr_seq2))
        diff = np.diff(fpr_seq3)
        self.our_auc = np.sum(diff*tpr_seq2)

        plt.figure()
        plt.plot(fpr_seq2,tpr_seq2,label = 'own function')
        plt.xlim([0,1])
        plt.ylim([0,1])
        plt.xlabel('False Positive Rate',fontsize=10)
        plt.ylabel('True Positive Rate',fontsize=10)
        plt.xticks(fontsize=10)
        plt.yticks(fontsize=10)
        plt.legend()
        plt.title('ROC curve',fontsize=10)

clf = linear_model.LogisticRegression(C=1e2,fit_intercept=True)
clf.fit(X_train,y_train)
y_lr_pred = clf.predict(X_train)
y_lr_pred = clf.predict(X_test)
y_lr_pred_prob = clf.predict_proba(X_test)
#F1
test_met = our_metrics(y_test, y_lr_pred, y_lr_pred_prob[:,1])
test_met.confusion_matrix()
print('our F1',test_met.F1)
#ROC
test_met.ROC()
#AUC
print('our AUC',test_met.our_auc)
```

```
from sklearn import metrics
y_score = clf.decision_function(X_test)
fpri, tpri, _ = metrics.roc_curve(y_test, y_score)
plt.plot(fpri,tpri,label = 'function from
    linear_model.LogisticRegression')
plt.legend()
plt.savefig("cla-ROC.png")
#AUC
roc_auc = metrics.auc(fpri, tpri)
print('their AUC',roc_auc)
```

6 局部建模

6.1 推导三阶样条的化简形式(6.1.2)。

$$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \theta_1 (x - \xi)_+^3 \tag{6.1.2}$$

答:

$$y = \begin{cases} \beta_{10} + \beta_{11} x + \beta_{12} x^2 + \beta_{13} x^3 & x \le \xi \\ \beta_{20} + \beta_{21} x + \beta_{22} x^2 + \beta_{23} x^3 & x \ge \xi \end{cases}$$

三个条件：

①连续

$\beta_{10} + \beta_{11}\xi + \beta_{12}\xi^2 + \beta_{13}\xi^3 = \beta_{20} + \beta_{21}\xi + \beta_{22}\xi^2 + \beta_{23}\xi^3$

②一阶导连续

$\beta_{11} + 2\beta_{12}\xi + 3\beta_{13}\xi^2 = \beta_{21} + 2\beta_{22}\xi + 3\beta_{23}\xi^2$

③二阶导连续

$2\beta_{12} + 6\beta_{13}\xi = 2\beta_{22} + 6\beta_{23}\xi$

$f(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \theta_1 (x - \xi)_+^3$

由③有：$3(\beta_{23} - \beta_{13})\xi = \beta_{12} - \beta_{22}$ ③'

由②有：$3(\beta_{23} - \beta_{13})\xi^2 + 2(\beta_{22} - \beta_{12})\xi = \beta_{11} - \beta_{21}$ ②'

由①有：$(\beta_{23} - \beta_{13})\xi^3 + (\beta_{22} - \beta_{12})\xi^2 + (\beta_{21} - \beta_{11})\xi = \beta_{10} - \beta_{20}$ ①'

不妨令 $\beta_{23} - \beta_{13} = \theta$。

代入③'，有 $\beta_{12} - \beta_{22} = 3\theta\xi$

代入②'，有 $\beta_{11} - \beta_{21} = 3\theta\xi^2 - 6\theta\xi^2 = -3\theta\xi^2$

代入①'，有 $\beta_{10} - \beta_{20} = \theta\xi^3 + (-3\theta\xi) \cdot \xi^2 + 3\theta\xi^3 = \theta\xi^3$

再令 $\begin{cases}\beta_{10}=\beta_0\\ \beta_{11}=\beta_1\\ \beta_{12}=\beta_2\\ \beta_{13}=\beta_3\end{cases}$，可推出 $\begin{cases}\beta_{20}=\beta_0-\theta\xi^3\\ \beta_{21}=\beta_1+3\theta\xi^2\\ \beta_{22}=\beta_2-3\theta\xi\\ \beta_{23}=\beta_3+\theta\end{cases}$

当 $x\geq\xi$ 时，

$$\begin{aligned}f(x)&=\beta_0+\beta_1x+\beta_0x^2+\beta_3x^3-\theta\xi^3+3\theta\xi^3x-3\theta\xi x^2+\theta x^3\\&=\beta_0+\beta_1x+\beta_0x^2+\beta_3x^3+\theta(x-\xi)^3\end{aligned}$$

当 $x\leq\xi$ 时，

$$f(x)=\beta_0+\beta_1x+\beta_0x^2+\beta_3x^3$$

综上：

$$f(x)=\beta_0+\beta_1x+\beta_0x^2+\beta_3x^3+\theta(x-\xi)_+^3$$

6.2 编写自然三阶样条的模拟仿真程序。

```
# 载入库
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la

# 自然三阶样条
def nature_cubic_spline(x,y,xi):
    n = np.size(x)
    d={}
    K = len(xi)
    for j in range(K-1):
        d[j] = ((x -
            xi[j])**3*(x-xi[j]>0)-(x-xi[K-1])**3*(x-xi[K-1]>0))/(xi[K-1]-xi[j])
    c = np.ones((n,1))
    X = np.hstack((c,x))
    for j in range(K-2):
        X = np.hstack((X,d[j]-d[K-2]))
    beta = la.inv(X.T.dot(X)).dot(X.T).dot(y)
    yhat = X.dot(beta)
    rk = x.ravel().argsort()
    return yhat,rk
```

```
# 生成数据
n = 100
x = np.random.rand(n,1)
error = np.random.randn(n,1)*0.3
y = np.sin(2*np.pi*x) + error
# 自然三阶样条估计
xi = np.linspace(1/5,4/5,4)
yhat,rk=nature_cubic_spline(x,y,xi)
plt.plot(x,y,'bo')
plt.plot(x[rk],yhat[rk],'r-')
```

6.3 编写平滑样条的模拟仿真程序。

```
# 生成模拟数据
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
n = 100
x = np.random.rand(n,1)
error = np.random.randn(n,1)*0.3
y = np.sin(2*np.pi*x) + error
plt.plot(x,y,'o')

# 平滑样条
xi = x.copy()
#xi = np.array([1/5,2/5,3/5,4/5])
# 计算B
def dk(xi,k,x):
    dk =
        ((x-xi[k])**3*(x-xi[k]>0)-(x-xi[-1])**3*(x-xi[-1]>0))/(xi[-1]-xi[k])
    return dk
p = xi.shape[0] - 1
B = np.zeros((n,p-1))
for j in range(p-1):
    print(j)
    B[:,j] =  dk(xi,j,x).ravel() - dk(xi,-2,x).ravel()
c = np.ones((n,1))
B = np.hstack((c,x,B))
# 计算 di''dj'' 的积分
def inte_dij_diff2(xi,i,j,uu):
```

```
    K = xi.shape[0]-1
    def inte_xiij(i,j,d,u):
        fu = u**3/3-(xi[i]+xi[j])*u**2/2+xi[i]*xi[j]*u
        fd = d**3/3-(xi[i]+xi[j])*d**2/2+xi[i]*xi[j]*d
        return fu-fd
    re = 36*(inte_xiij(i,j,xi[j],uu)+inte_xiij(K,K,xi[K],uu)
        -inte_xiij(i,K,xi[K],uu)-inte_xiij(j,K,xi[K],uu))
        /(xi[K]-xi[i])/(xi[K]-xi[j])
    return re
# 计算 Bi''Bj'' 的积分
def inte_Bij_diff2(xi,i,j,uu):
    K = xi.shape[0]-1
    re = inte_dij_diff2(xi,i,j,uu) + inte_dij_diff2(xi,K-1,K-1,uu)
        - inte_dij_diff2(xi,i,K-1,uu) - inte_dij_diff2(xi,j,K-1,uu)
    return re
# 生成二阶导矩阵
uu = 1
gamma_B = np.zeros((p+1,p+1))
for i in range(n-1):
    for j in range(i,p-1):
        print(i,j)
        gamma_B[i+2,j+2] = inte_Bij_diff2(xi,i,j,uu)
gamma_B = np.triu(gamma_B,1)+np.triu(gamma_B,0).T

lam1 = 0.01
lam2 = 0.05
lam3 = 0.1
yhat_lam1 = B.dot(la.inv(B.T.dot(B) + lam1 *
    gamma_B).dot(B.T).dot(y))
yhat_lam2 = B.dot(la.inv(B.T.dot(B) + lam2 *
    gamma_B).dot(B.T).dot(y))
yhat_lam3 = B.dot(la.inv(B.T.dot(B) + lam3 *
    gamma_B).dot(B.T).dot(y))
rk = x.ravel().argsort()
plt.figure(figsize = (16,9))
plt.title('Smoothing Spline',fontsize = 15)
plt.plot(x,y,'bo')
plt.plot(x[rk],yhat_lam1[rk],'r-', label = 'lambda = 0.001')
plt.plot(x[rk],yhat_lam2[rk],'g-', label = 'lambda = 0.05')
plt.plot(x[rk],yhat_lam3[rk],'m-', label = 'lambda = 1')
plt.legend(loc='lower left')
```

```
plt.show()
```

6.4 写出基于样条的非线性逻辑回归模型的估计方法，并进行简单的模拟仿真。

答: 基于样条的非线性逻辑回归模型：

$$log\frac{p(x;\beta)}{1-p(x;\beta)} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \sum_{k=1}^{K} \theta_k (x-\xi_k)_+^3$$

```
# 生成非线性数据
import numpy as np
import matplotlib.pyplot as plt
x1_1 = np.random.normal(-5,1,500)
x1_2 = np.random.normal(5,1,500)
x2_1 = np.random.normal(0,1,500)
x2_2 = np.random.normal(10,1,500)
X = np.hstack((x1_1,x1_2,x2_1,x2_2)).reshape(2000,1)
y1 = np.zeros((1000,1))
y2 = np.ones((1000,1))
y = np.vstack((y1,y2))
plt.figure()
plt.scatter(X,y)

# 普通logistic 回归方法失效
from sklearn import linear_model
clf = linear_model.LogisticRegression(C=1e2,fit_intercept=False)
clf.fit(X,y)
betax = clf.coef_.T
u = np.linspace(-7,12,2000)
fu = []
for uu in u:
    fuu = np.exp(uu*betax)/(1+np.exp(uu*betax))
    print(fuu)
    fu.append(fuu)
fu = np.asarray(fu).reshape(2000,1)
plt.figure()
plt.scatter(X[:1000],y1,color='r')
plt.scatter(X[1000:],y2,color='b')
plt.plot(u,fu)

# 三阶样条
```

```
xi1 = 1/3
xi2 = 2/3
k1 = (X-xi1)**3*(X-xi1>0)
k2 = (X-xi2)**3*(X-xi2>0)
c = np.ones((2000,1))
XX = np.hstack((c,X,X**2,X**3,k1,k2))

# 三阶样条logistic 回归
clf3 = linear_model.LogisticRegression(C=1e2,fit_intercept=False)
clf3.fit(XX,y)
betax3 = clf3.coef_.T
u = np.asarray(np.linspace(-7,12,2000)).reshape(2000,1)
U =
    np.hstack((c,u,u**2,u**3,(u-xi1)**3*(u-xi1>0),(u-xi2)**3*(u-xi2>0)))
fu3 = np.exp(U.dot(betax3))/(1+np.exp(U.dot(betax3)))
plt.figure()
plt.scatter(X[:1000],y1,color='r')
plt.scatter(X[1000:],y2,color='b')
plt.plot(u,fu3)
```

6.5 在例 6.2 中，对核技巧模拟仿真中的两个参数进行连续变化，观察最终拟合结果的变化。

答: 连续改变 tau 和 ld 的值。观察发现 tau 越小拟合曲线越光滑，ld 越大拟合曲线越平缓。

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la

n = 100
x = np.random.rand(n,1)
error = np.random.randn(n,1)*0.3
y = np.sin(2*np.pi*x)+error
rk = x.ravel().argsort()

tau = [0.01,0.1,1,10,100]
ld = [0.01,0.1,1,10,100]
fig = plt.figure(figsize=(10,10))
for i in range(len(tau)):
    for j in range(len(ld)):
```

```
        K = np.exp(-tau[i] *(x - x.T)**2)
        alpha = la.inv(K + ld[j]* np.eye(n)).dot(y)
        yhat2 = K.dot(alpha)
        plt.subplot(5,5,i+5*j+1)
        plt.figure()
        plt.plot(x,y,'ro',alpha=0.1)
        plt.plot(x[rk],yhat2[rk],'k-')
```

6.6 参考局部常数回归，写出 K 邻近估计法的目标函数。

答:

$$\sum_{i=1}^{N}(y_i - f(x))^2 I(x_i \in N_K(x))$$

6.7 自主下载 50ETF 期权合约一段时间的日收盘数据，画出每日的隐含分布，并结合市场走向观察隐含分布的变化。

答: 可从新浪财经等平台获取期权合约的日收盘数据，仿照案例 6.3.4 可画出每日的隐含分布。这里给出两天的实际数据以供参考练习。

```
#2021.8.2 的8 月看涨期权合约价格和行权价
call_price_1 = np.array([0.3450,0.2953,0.2474,0.1566,0.0816,
        0.0356,0.0130,0.0061,0.0034,0.0019,0.0013,0.0012])
call_K_1 = ([2.9,2.95,3.0,3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8,3.9])

#2021.8.3 的8 月看涨期权合约价格和行权价
call_price_2 = np.array([0.3577,0.3105,0.2604,0.1673,0.0895,
        0.0373,0.0126,0.0059,0.0031,0.0016,0.0011,0.0008])
call_K_2 = ([2.9,2.95,3.0,3.1,3.2,3.3,3.4,3.5,3.6,3.7,3.8,3.9])
```

6.8 写出非参数逻辑回归的局部似然估计的牛顿算法并进行模拟仿真。

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

# logistic 局部似然估计的牛顿算法
def local_likelihood_logistic(x,y,h,u,p):
    n = len(y)
    fu = []
    for u0 in u:
```

```
        X = np.ones((n,1))
        for i in range(1,p+1):
            X = np.hstack((X,(x-u0)**p))
        diff = 1
        iter = 0
        t = (x-u0)/h
        K = np.exp(-0.5*t**2)/h
        beta = la.pinv(X.T.dot(X)).dot(X.T).dot(y)
        while iter<1000 and diff>0.0001:
            like = np.sum((y*(X.dot(beta)) -
                np.log(1+np.exp(X.dot(beta))))*K)
            p_vec = np.exp(X.dot(beta))/(1+np.exp(X.dot(beta)))
            w = np.diag((p_vec*(1-p_vec)*K).ravel())
            beta = beta+la.pinv(X.T.dot(w).dot(X)).dot(X.T)
                .dot(np.diag((K).ravel())).dot(y - p_vec)
            like2 = np.sum((y*(X.dot(beta)) -
                np.log(1+np.exp(X.dot(beta))))*K)
            diff = np.abs(like - like2)
            iter = iter + 1
        fu.append(beta[0])
    fu = np.asarray(fu)
    return fu

# 生成非线性数据
x1_1 = np.random.normal(-5,1,100)
x1_2 = np.random.normal(5,1,100)
x2_1 = np.random.normal(0,1,100)
x2_2 = np.random.normal(10,1,100)
X = np.hstack((x1_1,x1_2,x2_1,x2_2)).reshape(400,1)
y1 = np.zeros((200,1))
y2 = np.ones((200,1))
y = np.vstack((y1,y2))
plt.figure()
plt.scatter(X,y)

# logistic 局部似然估计
p = 3
h = 0.1
u = np.asarray(np.linspace(-7,12,400)).reshape(400,1)
fu = local_likelihood_logistic(X,y,h,u,p)
plt.figure()
```

```
plt.scatter(X[:200],y1,color='r')
plt.scatter(X[200:],y2,color='b')
plt.plot(u,fu)
```

7 模型选择和模型评估

7.1 推导等式(7.1.7)。

$$LOOCV = \frac{1}{N}\sum_{i=1}^{N}[y_i - \hat{f}^{-i}(x_i)]^2 = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}\right]^2 \tag{7.1.7}$$

答:

$$\begin{aligned}\hat{f}^{-i} &= X\beta^{-i} \\ &= X(X^TX)^{-1}X^T(y + e_i(\hat{f}^{-i}(x_i) - y_i)) \\ &= S(y + e_i(\hat{f}^{-i}(x_i) - y_i))\end{aligned}$$

则去掉第 i 个样本后，第 i 个估计值为：

$$\hat{f}^{-i}(x_i) = e_i^TS(y + e_i(\hat{f}^{-i}(x_i) - y_i)) = e_i^TSy + S_{ii}(\hat{f}^{-i}(x_i) - y_i)$$

有如下推导：

$$\begin{aligned}\hat{f}^{-i}(x_i) - y_i &= e_i^TSy + S_{ii}(\hat{f}^{-i}(x_i) - y_i) - y_i \\ (1 - S_{ii})(\hat{f}^{-i}(x_i) - y_i) &= e_i^TSy - y_i \\ y_i - \hat{f}^{-i}(x_i) &= \frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}\end{aligned}$$

因此，可证明

$$LOOCV = \frac{1}{N}\sum_{i=1}^{N}\left[y_i - \hat{f}^{-i}(x_i)\right]^2 = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{y_i - \hat{f}(x_i)}{1 - S_{ii}}\right]^2$$

7.2 编写模拟仿真程序，使用交叉验证法分别选择岭回归和 Lasso 的惩罚参数 λ。

```
# 生成数据
n = 100
beta_true = np.array([1,-2,3,-4,5,-6,7,-8])
p = len(beta_true)
X = np.random.randn(n,p)
error = np.random.randn(n,1)*0.3
y = X.dot(beta_true.reshape(p,1))+error

# CV
from sklearn import linear_model
```

```
from sklearn.model_selection import KFold
z = np.hstack((X,y))
kf = KFold(n_splits=10)
## cross-validation
cv_seq_lasso = []
cv_seq_ridge = []
lambda_seq = np.linspace(-10,100,500)
for ld in lambda_seq:
    cv_lasso = 0
    cv_ridge = 0
    for train, test in kf.split(z):
    train_X = X[train]
    train_y = y[train]
    test_X = X[test]
    test_y = y[test]
    #lasso
    reg_lasso =
        linear_model.Lasso(alpha=ld,normalize=True,fit_intercept=False)
    reg_lasso.fit(train_X,train_y)
    y_lasso = reg_lasso.predict(test_X)
    cv_lasso = cv_lasso + np.mean((test_y.ravel() - y_lasso)**2)
    #岭 回 归
    reg_ridge =
        linear_model.Ridge(alpha=ld,normalize=True,fit_intercept=False)
    reg_ridge.fit(train_X,train_y)
    y_ridge = reg_ridge.predict(test_X)
    cv_ridge = cv_ridge + np.mean((test_y.ravel() - y_ridge)**2)
    cv_seq_lasso.append(cv_lasso/10)
    cv_seq_ridge.append(cv_ridge/10)

# 输 出 结 果
print('lasso:',lambda_seq[np.argmin(cv_seq_lasso)])
print('ridge:',lambda_seq[np.argmin(cv_seq_ridge)])
```

7.3 推导偏差-方差分解式 (7.2.1)。

答:

$$
\begin{aligned}
Err_x &= E\left\{[Y-\hat{f}(x)]^2|X=x\right\} \\
&= E\left\{[f(X)+\epsilon-\hat{f}(x)]^2|X=x\right\} \\
&= E[\epsilon|X=x]^2 + E\left\{[f(X)-\hat{f}(x)]^2|X=x\right\} + 2E[\epsilon[f(X)-\hat{f}(x)]|X=x] \\
&= \sigma_\epsilon^2 + E\left\{[f(X)-E\hat{f}(x)+E\hat{f}(x)-\hat{f}(x)]^2|X=x\right\} + 0 \\
&= \sigma_\epsilon^2 + E\left\{[f(X)-E\hat{f}(x)]^2|X=x\right\} + E\left\{[E\hat{f}(x)-\hat{f}(x)]^2|X=x\right\} \\
&\quad + E\left\{2[f(X)-E\hat{f}(x)][E\hat{f}(x)-\hat{f}(x)]|X=x\right\} \\
&= \sigma_\epsilon^2 + [E\hat{f}(x)-f(x)]^2 + E[\hat{f}(x)-E\hat{f}(x)]^2 \\
&= \sigma_\epsilon^2 + Bias^2[\hat{f}(x)] + Var[\hat{f}(x)]
\end{aligned}
$$

7.4 编写模拟仿真程序，分别使用 AIC、BIC 和 AICc 选择岭回归的惩罚参数 λ。

```
import numpy as np
import numpy.linalg as la
from sklearn import linear_model

# 生成数据
n = 100
beta_true = np.array([1,-2,3,-4,5,-6,7,-8])
p = len(beta_true)
X = np.random.randn(n,p)
error = np.random.randn(n,1)*0.3
y = X.dot(beta_true.reshape(p,1))+error

# 准则
AIC_seq_lasso = []
AIC_seq_ridge = []
BIC_seq_lasso = []
BIC_seq_ridge = []
AICc_seq_lasso = []
AICc_seq_ridge = []
lambda_seq = np.linspace(1,100,500)
for ld in lambda_seq:
    #lasso
    reg_lasso = linear_model.Lasso(alpha=ld,fit_intercept=False)
    reg_lasso.fit(X,y)
```

```
    y_lasso = reg_lasso.predict(X)
    beta_lasso = reg_lasso.coef_
    err_lasso = np.mean((y - y_lasso)**2)
    Sigma = np.diag((np.sign(beta_lasso[beta_lasso!=0])
        *ld/np.abs(beta_lasso[beta_lasso!=0])))
    XX = X[:,beta_lasso!=0]
    df_lasso =
        np.trace(XX.dot(la.pinv(XX.T.dot(XX)+n*Sigma)).dot(XX.T))
    AIC_lasso = err_lasso + 2*df_lasso*err_lasso/n
    AIC_seq_lasso.append(AIC_lasso)
    BIC_lasso = err_lasso + np.log(n)*df_lasso*err_lasso/n
    BIC_seq_lasso.append(BIC_lasso)
    AICc_lasso = AIC_lasso + err_lasso +
        (2*df_lasso*err_lasso/n)*(n/(n-df_lasso-1))
    AICc_seq_lasso.append(AICc_lasso)
    #岭 回 归
    reg_ridge = linear_model.Ridge(alpha=ld,fit_intercept=False)
    reg_ridge.fit(X,y)
    y_ridge = reg_ridge.predict(X)
    beta_ridge = reg_lasso.coef_
    err_ridge = np.mean((y - y_ridge)**2)
    df_ridge =
        np.trace(X.dot(la.pinv(X.T.dot(X)+n*np.eye(8))).dot(X.T))
    AIC_ridge = err_ridge + 2*df_ridge*err_ridge/n
    AIC_seq_ridge.append(AIC_ridge)
    BIC_ridge = err_ridge + np.log(n)*df_ridge*err_ridge/n
    BIC_seq_ridge.append(BIC_ridge)
    AICc_ridge = AIC_ridge + err_ridge +
        (2*df_ridge*err_ridge/n)*(n/(n-df_ridge-1))
    AICc_seq_ridge.append(AICc_ridge)

# 输 出 结 果
print('lasso_AIC:',lambda_seq[np.argmin(AIC_seq_lasso)])
print('lasso_BIC:',lambda_seq[np.argmin(BIC_seq_lasso)])
print('lasso_AICs:',lambda_seq[np.argmin(AICc_seq_lasso)])
print('ridge_AIC:',lambda_seq[np.argmin(AIC_seq_ridge)])
print('ridge_BIC:',lambda_seq[np.argmin(BIC_seq_ridge)])
print('ridge_AICs:',lambda_seq[np.argmin(AICc_seq_ridge)])
```

7.5 写出主成分回归的的有效自由度。

答: 主成分回归的有效自由度 = 选取主成分的个数。

取前 q 个主成分做回归，令 $Z_{n\times q} = X_{n\times p}U_{p\times q}$，U 每列是正交的。

有 $\hat{y} = \mathbf{S}y = Z(Z^TZ)^{-1}Z^Ty$，

则主成分回归的有效自由度为：

$$trace(\mathbf{S}) = trace(Z(Z^TZ)^{-1}Z^T) = trace((Z^TZ)^{-1}(Z^TZ)) = trace(I_q) = q$$

7.6 设 $\hat{\mathbf{y}} = \mathbf{S}\mathbf{y}$，证明 $\text{trace}(\mathbf{S}) = \sum_{i=1}^{N}\text{Cov}(\hat{y}_i, y_i)/\sigma_\varepsilon^2$。

答:

$$Cov(\hat{y}, y) = Cov(\mathbf{S}y, y) = \mathbf{S}Cov(y, y)$$

$$trace(Cov(\hat{y}, y)) = trace(\mathbf{S}Cov(y, y)) = \sigma_\epsilon^2 trace(\mathbf{S})$$

$$trace(\mathbf{S}) = \frac{trace(Cov(\hat{y}, y))}{\sigma_\epsilon^2} = \frac{\sum_{i=1}^{N}Cov(\hat{y}_i, y_i)}{\sigma_\epsilon^2}$$

8 统计推断基础

8.1 在线性回归模型式(8.1.6)中，如果误差项来自拉普拉斯分布，求 $\boldsymbol{\beta}$ 的极大似然估计。

$$Y = \mathbf{x}^T\boldsymbol{\beta} + \epsilon \tag{8.1.6}$$

答:

$$Y = X^T\beta + \varepsilon,\ \varepsilon \sim Laplace(0, b)$$

则 $f(\varepsilon) = \frac{1}{2b}e^{-\frac{|\varepsilon|}{2b}}$，可以写出似然函数为

$$L(\beta) = \prod_{i=1}^{N} \frac{1}{2b} e^{-\frac{\left|y_i - x_i^T\beta\right|}{2b}}$$

$$\ell(\beta) = -\sum_{i=1}^{N} log(2b) - \frac{1}{2b}\sum_{i=1}^{N}\left|y_i - x_i^T\beta\right|$$

可得到极大似然估计

$$\hat{\beta} = argmin \sum_{i=1}^{N}\left|y_i - x_i^T\beta\right|$$

8.2 在逻辑回归模型中，对似然比检验进行模拟仿真，并画出检验的功效函数。

```
import numpy as np
import numpy.linalg as la
from scipy.stats import chi2

np.random.seed(123)
x1 = np.random.random(50)
x2 = np.random.random(50)
z = -2.5 + 3*x1 + 4*x2
p = 1/(1+np.exp(-z))
y = np.random.binomial(1,p,50)
X = np.vstack((np.ones(50),x1,x2)).T

class logistic:
    def __init__(self,X,y):
        self.X = X
        self.y = y

    def fit(self,beta):
```

```
        X = self.X
        y = self.y
        diff = 1
        itera = 0
        while (itera<1000) & (diff>0.0001):
            like = np.sum(y*(X.dot(beta)) -
                np.log(1+np.exp(X.dot(beta))))
            p = 1/(1+np.exp(-X.dot(beta)))
            w = np.diag(p*(1-p))
            z = X.dot(beta) + la.inv(w).dot(y-p)
            beta =
                la.pinv(X.T.dot(w).dot(X)).dot(X.T).dot(w).dot(z)
            like2 = np.sum(y*(X.dot(beta)) -
                np.log(1+np.exp(X.dot(beta))))
            diff = np.abs(like - like2)
            itera += 1
        return beta

beta1 = logistic(X,y).fit(np.array([1,1,1]))
beta2 = logistic(X[:,:2],y).fit(np.array([1,1]))
like1 = np.sum(y*(X.dot(beta1)) - np.log(1+np.exp(X.dot(beta1))))
like2 = np.sum(y*(X[:,:2].dot(beta2)) -
    np.log(1+np.exp(X[:,:2].dot(beta2))))
LR = np.exp(like1-like2)
```

8.3 参考例 8.1，编写 Residual Bootstrap 模拟仿真程序。

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt

# 生成数据
n = 100
x = np.random.rand(n,1)
err = np.random.standard_t(3,(n,1))*0.4
y = np.sin(2*np.pi*x) + err

# 参数bootstrape
B = 1000
xi1 = 0.33
xi2 = 0.67
```

```
c = np.ones((n,1))
u = np.linspace(0,1,100).reshape(100,1)
cu = np.ones((100,1))
xiu1 = (u - xi1)**3*(u - xi1>0)
xiu2 = (u - xi2)**3*(u - xi2>0)
U = np.hstack((cu,u,u**2,u**3,xiu1,xiu2))

# Residual Bootsrape
k1 = ((x - xi1)*(x - xi1>0))**3
k2 = ((x - xi2)*(x - xi2>0))**3
X = np.hstack((c,x,x**2,x**3,k1,k2))
beta_ols = la.inv(X.T.dot(X)).dot(X.T).dot(y)
yhat = X.dot(beta_ols)
# Residual Bootstrape
residual = y - yhat
estf = np.zeros((100,B))
for j in range(B):
    nx = x
    id = np.random.randint(0,100,(100,))
    ny = yhat + residual[id]
    k1 = ((nx - xi1)*(nx - xi1>0))**3
    k2 = ((nx - xi2)*(nx - xi2>0))**3
    X = np.hstack((c,nx,nx**2,nx**3,k1,k2))
    beta = la.inv(X.T.dot(X)).dot(X.T).dot(ny)
    est = U.dot(beta)
    estf[:,j] = est.ravel()

# 模拟仿真结果
qt975 = np.percentile(estf,97.5,axis = 1)
qt025 = np.percentile(estf,2.5,axis = 1)
qt050 = np.percentile(estf,50,axis = 1)
plt.plot(x,y,'bo')
plt.plot(u,qt975,'r--')
plt.plot(u,qt025,'--')
plt.plot(u,qt050,'k-')
```

8.4 使用 Bootstrap 方法求逻辑回归的置信区间。

```
import numpy as np
import numpy.linalg as la
```

```
np.random.seed(123)
x1 = np.random.random(50)
x2 = np.random.random(50)
z = -2.5 + 3*x1 + 4*x2
p = 1/(1+np.exp(-z))
y = np.random.binomial(1,p,50)
X = np.vstack((np.ones(50),x1,x2)).T

class logistic:
    def __init__(self,X,y):
        self.X = X
        self.y = y

    def fit(self):
        X = self.X
        y = self.y
        beta = np.array([1,1,1])
        diff = 1
        itera = 0
        while (itera<1000) & (diff>0.0001):
            like = np.sum(y*(X.dot(beta)) -
                np.log(1+np.exp(X.dot(beta))))
            p = 1/(1+np.exp(-X.dot(beta)))
            w = np.diag(p*(1-p))
            z = X.dot(beta) + la.inv(w).dot(y-p)
            beta =
                la.pinv(X.T.dot(w).dot(X)).dot(X.T).dot(w).dot(z)
            like2 = np.sum(y*(X.dot(beta)) -
                np.log(1+np.exp(X.dot(beta))))
            diff = np.abs(like - like2)
            itera += 1
        return beta

    def bootstrap(n,X,y):
        length = len(y)
        betalist = []
        for _ in range(n):
            idx =
                np.random.choice(np.arange(length),size=length,replace=True)
            mylogi = logistic(X[idx],y[idx])
            beta = mylogi.fit()
```

```
            betalist.append(beta[1])
        return np.array(betalist)

betas = bootstrap(500,X,y)
left = np.percentile(betas,0.05)
right = np.percentile(betas,0.9)
```

8.5 使用 Bootstrap 方法求岭回归和 Lasso 参数的置信区间，其中惩罚参数 λ 由交叉验证法选取。

```
# 以岭回归为例，lasso类似，把岭回归的拟合部分换成Lasso方法即可。
def bootstrap_confidence_intervel_ridge_regression():
    def cv(K,X,y):
        z = np.hstack((X,y))
        kf = KFold(n_splits=K)
        train_seq = []
        test_seq = []
        for train,test in kf.split(z):
            train_seq.append(train)
            test_seq.append(test)

        lam_set = np.linspace(0.1,10,100)
        cv_seq_r = []
        for lam in lam_set:
            cv_r = 0
            for j in range(K):
                train_x = X[train_seq[j]]
                train_y = y[train_seq[j]]
                test_x = X[test_seq[j]]
                test_y = y[test_seq[j]]
                model_ridge = Ridge(alpha=lam,fit_intercept= True)
                beta_h_r = model_ridge.fit(train_x,train_y)
                est_r = test_x.dot((beta_h_r.coef_).T)
                cv_r = cv_r + np.mean((test_y.ravel() -
                    est_r.ravel())**2)
            cv_seq_r.append(cv_r/K)
        blam = lam_set[np.argmin(cv_seq_r)]
        model_ridge = Ridge(alpha=blam,fit_intercept= True)
        beta_br = model_ridge.fit(train_x,train_y)
        return beta_br.coef_
```

```
    # data
    n = 100
    x1 = np.random.randn(n,1)
    x2 = np.random.randn(n,1)
    error = np.random.randn(n,1)*0.3
    y = 1-5*x1+0.01*x2+error
    X = np.hstack((x1,x1,x2))

    K = 5
    B = 200
    rset = np.zeros((B,3))
    for j in range(B):
        id = np.random.randint(0,100,(100,))
        nx = X[id]
        ny = y[id]
        r = cv(K,nx,ny)
        rset[j,:] = r.ravel()
    betaqt975 = np.percentile(rset,97.5,axis = 0)
    betaqt025 = np.percentile(rset,2.5,axis = 0)
    print("beta1的95%置信区间为: ["+str(betaqt025[0])+",
        "+str(betaqt975[0])+"]")
    print("beta2的95%置信区间为: ["+str(betaqt025[1])+",
        "+str(betaqt975[1])+"]")
    print("beta3的95%置信区间为: ["+str(betaqt025[2])+",
        "+str(betaqt975[2])+"]")
bootstrap_confidence_intervel_ridge_regression()
```

8.6 推导 $KL(p||q) \geq 0$。

答: 根据 Jensen 不等式，

$$\begin{aligned} KL(p||q) &= \int p(x) log \frac{p(x)}{q(x)} dx = -\int p(x) log \frac{q(x)}{p(x)} dx \\ &\geq -log \int p(x) \frac{q(x)}{p(x)} dx = -log \int q(x) dx = 0 \end{aligned}$$

8.7 验证式 (8.5.5)，并证明 EM 算法的单调上升性质。

答: 似然函数：

$$l(\theta^k|x) = \int q(z|x,\theta^k) logp(x,z|\theta^k)dz - \int q(z|x,\theta^k) logq(z|x,\theta^k)dz$$
$$l(\theta^{k+1}|x) = \int q(z|x,\theta^k) logp(x,z|\theta^{k+1})dz - \int q(z|x,\theta^k) logq(z|x,\theta^{k+1})dz$$

似然函数的第一项，根据 Jensen 不等式有：

$$\begin{aligned}
&\int q(z|x,\theta^k) logq(z|x,\theta^{k+1})dz - \int q(z|x,\theta^k) logq(z|x,\theta^k)dz \\
&= \int q(z|x,\theta^k) log\frac{q(z|x,\theta^{k+1})}{q(z|x,\theta^k)}dz \\
&\leq log \int q(z|x,\theta^k)\frac{q(z|x,\theta^{k+1})}{q(z|x,\theta^k)}dz \\
&\leq log \int q(z|x,\theta^{k+1})dz = 0
\end{aligned}$$

似然函数的第二项记为：

$$Q(\theta|\theta^k) = \int q(z|x,\theta^k) logp(z|x,\theta)dz$$

在 E 步中，计算 Q 函数；在 M 步中，迭代规则为：

$$\theta^{k+1} = argmax Q(\theta|\theta^k)$$

因此有：

$$Q(\theta^{k+1}|\theta^k) \geq Q(\theta^{k+1}|\theta^k)$$

综上，证明了 EM 的单调性：

$$l(\theta^{k+1}|x) \geq l(\theta^k|x)$$

8.8 在 EM 算法中，推导式(8.5.14)和式(8.5.16)。

$$r_{ik} = \frac{\hat{\pi}_k \phi(\mathbf{x}_i; \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{l=1}^{K} \hat{\pi}_l \phi(\mathbf{x}_i; \hat{\mu}_l, \hat{\Sigma}_l)} \tag{8.5.14}$$

$$\hat{\pi}_k = \frac{1}{N}\sum_{i=1}^{N} r_{ik} \tag{8.5.16}$$

答: 推导式(8.5.14):

$$\begin{aligned}
r_{ik} &= E(z_{ik}|\mathbf{x}_i, \hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k) \\
&= P(z_{ik} = 1|\mathbf{x}_i, \hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k) \\
&= \frac{P(z_{ik} = 1, \mathbf{x}_i|\hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k)}{P(\mathbf{x}_i)} \\
&= \frac{P(z_{ik} = 1, \mathbf{x}_i|\hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k)}{\sum_{l=1}^{K} P(z_{il} = 1, \mathbf{x}_i|\hat{\mu}_l, \hat{\Sigma}_l, \hat{\pi}_l)} \\
&= \frac{P(\mathbf{x}_i|z_{ik} = 1, \hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k) \cdot P(z_{ik} = 1|\hat{\mu}_k, \hat{\Sigma}_k, \hat{\pi}_k)}{\sum_{l=1}^{K} P(\mathbf{x}_i|z_{il} = 1, \hat{\mu}_l, \hat{\Sigma}_l, \hat{\pi}_l) \cdot P(z_{il} = 1|\hat{\mu}_l, \hat{\Sigma}_l, \hat{\pi}_l)} \\
&= \frac{\hat{\pi}_k \phi(\mathbf{x}_i; \hat{\mu}_k, \hat{\Sigma}_k)}{\sum_{l=1}^{K} \hat{\pi}_l \phi(\mathbf{x}_i; \hat{\mu}_l, \hat{\Sigma}_l)}
\end{aligned}$$

推导式(8.5.16)：

由于有约束 $\sum_{k=1}^{K} \pi_k = 1$，可以构造拉格朗日函数 $Q_1(\theta)$ 然后求解。

$$Q_1(\theta) = \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik} log \pi_k + \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik} log \phi(x_i|\mu_k, \Sigma_k) + \lambda(\sum_{k=1}^{K} \pi_k - 1)$$

分别对 π_k 和 λ 求导：

$$\frac{\partial Q_1(\theta)}{\partial \pi_k} = \sum_{i=1}^{N} \frac{r_{ik}}{\pi_k} + \lambda = 0$$

$$\frac{\partial Q_1(\theta)}{\partial \lambda} = \sum_{k=1}^{K} \pi_k - 1 = 0$$

由 (13) 式可得 $\pi_k = \frac{\sum_{i=1}^{N} r_{ik}}{\lambda}$, 对 k 求和并与 (14) 式联立可得：

$$\sum_{k=1}^{K} \pi_k = \frac{\sum_{k=1}^{K}\sum_{i=1}^{N} r_{ik}}{\lambda} = 1$$

因此可知 $\lambda = \sum_{k=1}^{K}\sum_{i=1}^{N} r_{ik} = N$，于是 $\hat{\pi}_k = \frac{\sum_{i=1}^{N} r_{ik}}{N}$。

8.9 编写混合多元正态模型的 EM 算法。

```
from __future__ import print_function
import numpy as np

def generateData(k,mu,sigma,dataNum):
    ''' 产生混合高斯模型的数据 '''
    # 初始化数据
    dataArray = np.zeros(dataNum,dtype=np.float32)
    # 逐个依据概率产生数据
    # 高斯分布个数
    n = len(k)
    for i in range(dataNum):
        # 产生[0,1]之间的随机数
        rand = np.random.random()
        Sum = 0
        index = 0
        while(index < n):
            Sum += k[index]
            if(rand < Sum):
                dataArray[i] =
                    np.random.normal(mu[index],sigma[index])
                break
            else:
                index += 1
    return dataArray

def normPdf(x,mu,sigma):
    ''' 计算均值为mu ，标准差为sigma 的正态分布函数的密度函数值 '''
    return (1./np.sqrt(2*np.pi))*(np.exp(-(x-mu)**2/(2*sigma**2)))

def em(dataArray,k,mu,sigma,step = 10):
    ''' em 算法估计高斯混合模型 '''
    # 高斯分布个数
    n = len(k)
    # 数据个数
    dataNum = dataArray.size
    # 初始化gama 数组
    gamaArray = np.zeros((n,dataNum))
    for s in range(step):
        for i in range(n):
```

```
            for j in range(dataNum):
                Sum =
                    sum([k[t]*normPdf(dataArray[j],mu[t],sigma[t])
                    for t in range(n)])
                gamaArray[i][j] =
                    k[i]*normPdf(dataArray[j],mu[i],sigma[i])/float(Sum)
        # 更新 mu
        for i in range(n):
            mu[i] =
                np.sum(gamaArray[i]*dataArray)/np.sum(gamaArray[i])
        # 更新 sigma
        for i in range(n):
            sigma[i] = np.sqrt(np.sum(gamaArray[i]*(dataArray -
                mu[i])**2)/np.sum(gamaArray[i]))
        # 更新系数 k
        for i in range(n):
            k[i] = np.sum(gamaArray[i])/dataNum
    return [k,mu,sigma]

if __name__ == '__main__':
    # 参数的准确值
    k = [0.3,0.4,0.3]
    mu = [2,4,3]
    sigma = [1,1,4]
    # 样本数
    dataNum = 5000
    # 产生数据
    dataArray = generateData(k,mu,sigma,dataNum)
    # 参数的初始值
    # 注意em 算法对于参数的初始值是十分敏感的
    k0 = [0.3,0.3,0.4]
    mu0 = [1,2,2]
    sigma0 = [1,1,1]
    step = 6
    # 使用em 算法估计参数
    k1,mu1,sigma1 = em(dataArray,k0,mu0,sigma0,step)
    # 输出参数的值
    print("参数实际值:")
    print("k:",k)
    print("mu:",mu)
    print("sigma:",sigma)
```

```
    print("参数估计值:")
    print("k1:",k1)
    print("mu1:",mu1)
    print("sigma1:",sigma1)
```

9 贝叶斯方法

9.1 具体说明 Lasso 如何作为参数具有 Laplace 先验分布的后验众数。

答: 对线性模型 $y_i = w^T \mathbf{x}_i + \epsilon_i$, 其中 $\epsilon_i \sim N(0, 1/\beta)$ 。对参数 w 的分布加入先验分布信息 $w \sim \text{Laplace}\left(0, \frac{1}{\lambda}\right)$，则由

$$p(w \mid x, y) = \frac{p(y \mid x, w) \cdot p(w)}{p(y)}$$

可以得到 MAP 方程:

$$\begin{aligned}
\arg\max_w L(w) = & \text{ likelihiood } \times \text{ prior} \\
&= P(x, y \mid w) \times P(w) \\
&= \prod_{i=1}^{m} \sqrt{\frac{\beta}{2\pi}} \exp\left(\frac{-\beta\left(y_i - w^T \mathbf{x}_i\right)^2}{2}\right) \cdot \prod_{j=1}^{n} \frac{\lambda}{2} \exp\left(-\lambda \left|w_j\right|\right)
\end{aligned}$$

取对数得:

$$\begin{aligned}
\arg\max_w \ell(w) &= \ln L(w) \\
&= \ln\left[\prod_{i=1}^{m} \sqrt{\frac{\beta}{2\pi}} \exp\left(\frac{-\beta\left(y_i - w^T \mathbf{x}_x\right)^2}{2}\right) \cdot \prod_{j=1}^{n} \frac{\lambda}{2} \exp\left(-\lambda \left|w_j\right|\right)\right] \\
&= \ln \prod_{i}^{m} + \ln \prod_{j}^{n} \\
&= \sum_{i}^{m} \ln + \sum_{i}^{n} \ln \\
&= \sum_{i}^{m} \left[\frac{1}{2} \ln \frac{\beta}{2\pi} - \frac{\beta}{2}\left(y_i - w^T \mathbf{x}_i\right)^2\right] + \sum_{j}^{n} \left[\ln \frac{\lambda}{2} - \lambda \left|w_j\right|\right] \\
&= -\frac{g}{2} \sum_{i}^{m} \left(y_i - w^T \mathbf{x}_i\right)^2 - \lambda \sum_{j}^{n} \left|w_j\right| + \sum_{i}^{m} \frac{1}{2} \ln \frac{\beta}{2\pi} + \sum_{j}^{n} \ln \frac{\lambda}{2} \\
&= -\frac{\beta}{2} \sum_{i}^{m} \left(y_i - w^T \mathbf{x}_i\right)^2 - \lambda \sum_{i}^{n} \left|w_j\right| + \text{ constant}
\end{aligned}$$

等价于:

$$\begin{aligned}
\arg\min_= f(w) &= \sum_{1}^{m} \left(y_i - w^T \mathbf{x}_i\right)^2 + \lambda \sum_{j}^{n} \left|w_j\right| \\
&= (Y - XW)^2 + \lambda \|W\|_1 \\
&= |Y - XW|_2^2 + \lambda \|W\|_1
\end{aligned}$$

9.2 在拒绝抽样法中，证明 $P\left[X \leq x, U \leq \frac{h(X)}{Mq(X)}\right] = \frac{\int_{-\infty}^{x} f(x)dx}{cM}$。

答:

$$\begin{aligned} P\left(X \leq x, U \leq \frac{h(X)}{Mq(X)}\right) &= \int_{-\infty}^{+\infty} P\left(X \leq x, U \leq \frac{h(X)}{Mq(X)} \mid X = \omega\right) q(\omega)d\omega \\ &= \int_{-\infty}^{+\infty} P\left(U \leq \frac{h(X)}{Mq(X)}\right) I(X \leq x)q(X)dx \\ &= \int_{-\infty}^{x} P\left(U \leq \frac{h(X)}{Mq(X)}\right) q(X)dx \\ &= \frac{\int_{-\infty}^{x} f(x)dx}{cM} \end{aligned}$$

9.3 验证 Gibbs 抽样的接受率为 1。

答: 在 Gibbs 抽样中接受率表示为 $\frac{\pi(x^*)q\left(x_i|x^*_{-i}\right)}{\pi(x)q\left(x^*_i|x_{-i}\right)}$。

由于 x_i 和 x^*_{-i} 只有 i 不同, 但该符号表示去除 i, 因此 $x_i = x^*_{-i}$ 另外 gibbs sampling 基于假设各维度不相关, 可得以下关系

$$\pi\left(x^*\right) = \pi\left(x^*_i * \pi\left(x^*_{-i}\right)\right) = \pi\left(x^*_i \mid x^*_{-i}\right) * \pi\left(x^*_{-i}\right)$$

综上,

$$\frac{\pi\left(x^*\right) q\left(x_i \mid x^*_{-i}\right)}{\pi(x) q\left(x^*_i \mid x_{-i}\right)} = \frac{\pi\left(x^*_i \mid x^*_{-i}\right) * \pi\left(x^*_-\right) q\left(x_i \mid x^*_{-i}\right)}{\pi\left(x_i \mid x_{-i}\right) * \pi\left(x_{-i}\right) q\left(x^*_i \mid x_{-i}\right)} = 1$$

9.4 给出式 (9.3.24) 积分存在的条件。

答: $g(X)$ 二阶矩存在。

9.5 在例 9.3 中，如果参数 α、β 的先验分布变为 $U(-30, 30)$ 的均匀分布（τ 的先验分布不变），使用原来的先验分布下得到的 Gibbs 样本，利用重要性抽样得到新的先验分布对应的后验均值。

答: 比例重要性抽样估计

$$\hat{\mu} = \frac{\frac{1}{N}\sum_{i=1}^{N} h(X_i)g(X_i)/q(X_i)}{\frac{1}{N}\sum_{i=1}^{N} h(X_i)/q(X_i)}$$

目前已有样本的抽样分布为

$$\begin{aligned} posterior(\alpha,\beta,\tau|\mathbf{y},\mathbf{x}) &\propto L(\alpha,\beta,\tau|\mathbf{y},\mathbf{x}) \times \mathbf{prior}(\alpha,\beta,\tau) \\ &\propto \tau^{a+\frac{N}{2}-1}\exp\left\{-\frac{\tau}{b}-\frac{\tau}{2}\sum_{i=1}^{N}(y_i-\alpha-\beta x_i)^2\right\} \\ &\quad \times \exp\left\{-\frac{\tau_\alpha}{2}(\alpha-\mu_\alpha)^2-\frac{\tau_\beta}{2}(\beta-\mu_\beta)^2\right\}. \end{aligned}$$

每个参数的条件分布如下：

$$\begin{aligned} \alpha|\beta,\tau,\mathbf{x},\mathbf{y} &\sim N\left(\frac{\tau_\alpha\mu_\alpha+\tau\sum_i(y_i-\beta x_i)}{\tau_\alpha+\tau N}, 1/(\tau_\alpha+\tau N)\right) \\ \beta|\alpha,\tau,\mathbf{x},\mathbf{y} &\sim N\left(\frac{\tau_\beta\mu_\beta+\tau\sum_i(y_i-\alpha)x_i}{\tau_\beta+\tau\sum_i x_i^2}, 1/(\tau_\beta+\tau\sum_i x_i^2)\right) \\ \tau|\alpha,\beta,\mathbf{x},\mathbf{y} &\sim Gamma\left(a+\frac{N}{2}, b+\sum_i\frac{(y_i-\alpha-\beta x_i)^2}{2}\right) \end{aligned}$$

改变先验后的后验分布为

$$\begin{aligned} posterior^*(\alpha,\beta,\tau|\mathbf{y},\mathbf{x}) &\propto L(\alpha,\beta,\tau|\mathbf{y},\mathbf{x}) \times \mathbf{prior}(\alpha,\beta,\tau) \\ &\propto \tau^{a+\frac{N}{2}-1}\exp\left\{-\frac{\tau}{b}-\frac{\tau}{2}\sum_{i=1}^{N}(y_i-\alpha-\beta x_i)^2\right\} \times \frac{1}{60}\times\frac{1}{60}. \end{aligned}$$

每个参数的条件分布如下：

$$\begin{aligned} \alpha^*|\beta,\tau,\mathbf{x},\mathbf{y} &\sim N\left(\frac{\tau\sum_i(y_i-\beta x_i)}{\tau N}, 1/\tau N\right) \\ \beta^*|\alpha,\tau,\mathbf{x},\mathbf{y} &\sim N\left(\frac{\tau\sum_i(y_i-\alpha)x_i}{\tau\sum_i x_i^2}, 1/\tau\sum_i x_i^2\right) \\ \tau^*|\alpha,\beta,\mathbf{x},\mathbf{y} &\sim Gamma\left(a+\frac{N}{2}, b+\sum_i\frac{(y_i-\alpha-\beta x_i)^2}{2}\right) \end{aligned}$$

```
#在例9.9 后加如下代码
alpha = m1.alpha[-400:]
beta = m1.beta[-400:]
tau = m1.tau[-400:]
tau_alpha = tau_beta = 1
N = 1000

#原先验分布下的后验分布参数
```

```
alpha_inv_var = tau_alpha + tau[-1] * N
alpha_mu = (tau[-1] * np.sum(y - beta[-1] * x)) / alpha_inv_var
beta_inv_var = tau_beta + tau[-1] * np.sum(x * x)
beta_mu = (tau[-1] * np.sum((y - alpha[-1]) * x)) / beta_inv_var
a, b = m1.a, 1 / m1.b
#新先验分布下的后验分布参数
alpha_inv_var_new = tau[-1] * N
alpha_mu_new = (tau[-1] * np.sum(y - beta[-1] * x)) / alpha_inv_var
beta_inv_var_new = tau[-1] * np.sum(x * x)
beta_mu_new = (tau[-1] * np.sum((y - alpha[-1]) * x)) /
    beta_inv_var
a_new, b_new = 1, 1 / m1.b

#重要性采样估计后验均值
alpha_pdf_nu = st.norm.pdf(alpha, alpha_mu, 1 / alpha_inv_var**0.5)
alpha_pdf_de = st.norm.pdf(alpha, alpha_mu_new, 1 /
    alpha_inv_var_new**0.5)
beta_pdf_nu = st.norm.pdf(beta, beta_mu, 1 / beta_inv_var**0.5)
beta_pdf_de = st.norm.pdf(beta, beta_mu_new, 1 /
    beta_inv_var_new**0.5)
tau_pdf_nu = st.gamma.pdf(tau, a, b)
tau_pdf_de = st.gamma.pdf(tau, a_new, b_new)

post_mu_alpha = np.mean(alpha_pdf_nu*alpha/alpha_pdf_de)
post_mu_beta = np.mean(beta_pdf_nu*beta/beta_pdf_de)
post_mu_tau = np.mean(tau_pdf_nu*tau/tau_pdf_de)

print("alpha后验均值:", post_mu_alpha)
print("beta后验均值:", post_mu_beta)
print("tau后验均值:", post_mu_tau)
```

9.6 计算例子 9.3 和 9.4 中的蒙特卡洛标准误。

答: 例 9.3 中蒙特卡洛标准误定义为 $std(\hat{\mu}) = std[f(X)g(X)/q(X)]/\sqrt{N}$。例 9.4 中，假设得到的后验分布的抽样序列为 $\theta_1, \cdots, \theta_N$，$\theta_i \sim \pi(\theta|\mathbf{X})$。此时，后验均值的蒙特卡洛标准误是 $\hat{\sigma}_\theta/\sqrt{N}$。故代码如下

```
def Standard_error(sample):
std=np.std(sample,ddof=0)
standard_error=std/np.sqrt(len(sample))
return standard_error
```

```
#例 9.3
print('alpha的标准误为',Standard_error(m1.alpha[-400:]))
print('beta的标准误为',Standard_error(m1.beta[-400:]))

#例 9.4
mui = st.norm.pdf(X,0,1)*M
mu_std = Standard_error(mui)
print('标准误为',mu_std)
```

9.7 验证式(9.4.6)和式(9.4.8)。

$$\mathcal{L}(q^*(\theta),\mathbf{x}) = -KL\left[q_j(\theta_j)||\exp(p^*(\mathbf{x},\theta_j))\right] + const \tag{9.4.6}$$

$$q_j(\theta_j) \propto \exp\left(\int \log p(\mathbf{x},\theta)\prod_{k\neq j}\left[q_k(\theta_k)d\theta_k\right]\right) \tag{9.4.8}$$

答: 首先我们证明等式(9.4.6)。记 $\theta_{-\mathbf{j}} = (\theta_1,\cdots,\theta_{\mathbf{j-1}},\theta_{\mathbf{j+1}},\cdots,\theta_1)$，$q^*(\theta_{-\mathbf{j}}) = \prod_{\mathbf{k}\neq\mathbf{j}}\mathbf{q_k}(\theta_\mathbf{k})$，则有

$$\begin{aligned}
\mathcal{L}\left(q^*\left(\theta\right),\mathbf{x}\right) &= \int q^*\left(\theta\right)\log\frac{p\left(\mathbf{x},\theta\right)}{q^*\left(\theta\right)}d\theta \\
&= \int q\left(\theta_j\right)q\left(\theta_{-\mathbf{j}}\right)\left(\log p\left(\mathbf{x},\theta\right) - \log q\left(\theta_j\right) - \log q\left(\theta_{-\mathbf{j}}\right)\right)d\theta \\
&= \int q\left(\theta_j\right)q\left(\theta_{-\mathbf{j}}\right)\left(\log p\left(\mathbf{x},\theta\right) - \log q\left(\theta_j\right)\right)d\theta \\
&\quad - \int q\left(\theta_j\right)q\left(\theta_{-\mathbf{j}}\right)\log q\left(\theta_{-\mathbf{j}}\right)d\theta \\
&= \int q\left(\theta_j\right)\left(\int q\left(\theta_{-\mathbf{j}}\right)\log p\left(\mathbf{x},\theta\right)d\theta_{-\mathbf{j}} - \log\mathbf{q}\left(\theta_\mathbf{j}\right)\right)d\theta_j \\
&\quad - \int q\left(\theta_j\right)\int q\left(\theta_{-\mathbf{j}}\right)\log q\left(\theta_{-\mathbf{j}}\right)d\theta_{-\mathbf{j}}\mathbf{d}\theta_\mathbf{j} \\
&= \int q\left(\theta_j\right)\log\frac{\exp\left(\left\langle\log p\left(\mathbf{x},\theta\right)\right\rangle_{q(\mathbf{theta}_{-\mathbf{j}})}\right)}{q\left(\theta_j\right)}d\theta_j + c \\
&= -KL\left[q\left(\theta_j\right)\|\exp(p^*(\mathbf{x},\theta_j))\right] + c
\end{aligned}$$

最后一个等式由定义 $\exp(p^*(\mathbf{x},\theta_j)) := \exp\left(\left\langle\log p\left(\mathbf{x},\theta\right)\right\rangle_{q(\theta_{-\mathbf{j}})}\right)$ 可得，$\langle\cdot\rangle_{q(\theta_{-\mathbf{j}})}$ 是在后验分布 $q\left(\theta_{-\mathbf{j}}\right)$ 下关于 $\theta_{-\mathbf{j}}$ 的期望。即 $\exp(p^*(\mathbf{x},\theta_j)) = \exp\left(\int\log p^*(\mathbf{x},\theta)\prod_{k\neq j}\left[q_k(\theta_k)d\theta_k\right]\right)$ 最大化下界 $\mathcal{L}\left(q^*\left(\theta\right),\mathbf{x}\right)$，即等式(9.4.6)右边 KL 距离为 0 可得等式(9.4.8)。

9.8 计算简单均值模型的变分下界 $\mathcal{L}(q^*(\theta),\mathbf{x})$。

答:

$$\begin{aligned}\mathcal{L}\left(q^{*}\left(\theta\right),\mathbf{x}\right)=&\int q^{*}\left(\theta\right)\log\frac{p\left(\mathbf{x},\theta\right)}{q^{*}\left(\theta\right)}d\theta\\=&\langle\log p(\mathbf{x}\mid\theta)\rangle_{q^{*}(\theta)}+\langle\log p(\theta)\rangle_{q^{*}(\theta)}-\langle q^{*}(\theta)\rangle_{q^{*}(\theta)}\\=&\langle\log p(\mathbf{x}\mid\theta)\rangle_{q^{*}(\theta)}+\langle\log p(\mu\mid\tau)\rangle_{q^{*}(\theta)}+\langle\log p(\tau)\rangle_{q^{*}(\tau)}\\&-\langle\log q^{*}(\mu)\rangle_{q^{*}(\mu)}-\langle\log q^{*}(\tau)\rangle_{q^{*}(\tau)}\end{aligned}$$

其中,

$$\begin{aligned}\langle\log p(\mathbf{x}\mid\theta)\rangle_{q^{*}(\theta)}=&\frac{N}{2}\langle\log\tau\rangle_{q^{*}(\tau)}-\frac{N}{2}\log2\pi-\frac{\langle\tau\rangle_{q^{*}(\tau)}}{2}\langle(\mathbf{x}-\mu)^{T}(\mathbf{x}-\mu)\rangle_{q^{*}(\mu)},\\=&\frac{N}{2}(\psi(a_{N})-\log b_{N}-\log2\pi)-\frac{a_{N}}{2b_{N}}\left(\mathbf{x}^{T}\mathbf{x}-2\mu_{N}\mathbf{1}^{T}\mathbf{x}+N\left(\mu_{N}^{2}+\lambda_{N}\right)\right),\\\langle\log p(\mu\mid\tau)\rangle_{q^{*}(\theta)}=&\frac{1}{2}\langle\log\tau\rangle_{q^{*}(\tau)}-\frac{1}{2}\log2\pi/\lambda_{0}-\frac{\langle\tau\lambda_{0}\rangle_{q^{*}(\tau)}}{2}\langle(\mu-\rho_{0})^{2}\rangle_{q^{*}(\mu)},\\=&\frac{1}{2}(\psi(a_{N})-\log b_{N}-\log2\pi-\log\lambda_{0})-\frac{a_{N}}{2b_{N}}\left(\rho_{0}^{2}-2\rho_{0}\mu_{N}+\mu_{N}^{2}+\lambda_{N}\right),\\\langle\log p(\tau)\rangle_{q^{*}(\tau)}=&a_{0}\log b_{0}-\log\Gamma\left(a_{0}\right)+\left(a_{0}-1\right)\left(\langle\log\tau\rangle_{q^{*}(\tau)}\right)-b_{0}\langle\tau\rangle_{q^{*}(\tau)},\\=&a_{0}\log b_{0}-\log\Gamma\left(a_{0}\right)+\left(a_{0}-1\right)\left(\psi(a_{N})-\log b_{N}\right)-\frac{b_{0}a_{N}}{b_{N}},\\\langle\log q^{*}(\mu)\rangle_{q^{*}(\mu)}=&\frac{1}{2}\log\lambda_{N}-\frac{1}{2}\log2\pi-\frac{\lambda_{N}}{2}\langle(\mu-\rho_{0})^{2}\rangle_{q^{*}(\mu)},\\=&\frac{1}{2}\log\lambda_{N}-\frac{1}{2}\log2\pi-\frac{\lambda_{N}^{2}}{2},\\\langle\log q^{*}(\tau)\rangle_{q^{*}(\tau)}=&a_{N}\log b_{N}-\log\Gamma\left(a_{N}\right)+\left(a_{N}-1\right)\left(\langle\log\tau\rangle_{q^{*}(\tau)}\right)-b_{0}\langle\tau\rangle_{q^{*}(\tau)},\\=&a_{N}\log b_{N}-\log\Gamma\left(a_{N}\right)+\left(a_{N}-1\right)\left(\psi(a_{N})-\log b_{N}\right)-\frac{b_{N}a_{N}}{b_{N}},\\=&\left(a_{N}-1\right)\psi(a_{N})-\log\Gamma\left(a_{N}\right)-\log b_{N}-a_{N},\end{aligned}$$

10 树和树的集成

10.1 描述如何使用交叉验证法从树序列 $T_0 \supset T_1 \supset \cdots \supset T_r$ 中选取最优的子树。

答: 使用独立的验证集，计算各子树的平方误差或基尼系数，选择最小的作为最优决策树。

10.2 推导公式 (10.2.1)。

答: 样本为 $X = [x_1, x_2, ..., x_B]^T$，$cov(X) = \sigma^2 \begin{bmatrix} 1 & \rho & \rho & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \rho & \cdots & \cdots & 1 \end{bmatrix}$，

$$cov(\frac{1}{B}\mathbf{1}^T x) = \frac{1}{B}\mathbf{1}^T cov(x)\mathbf{1}\frac{1}{B} = \frac{1}{B^2}\mathbf{1}^T cov(x)\mathbf{1}$$

因为 $\mathbf{1}^T \begin{bmatrix} 1 & \rho & \cdots & \rho \\ \cdots & 1 & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 1 \end{bmatrix} \mathbf{1} = \begin{bmatrix} (B-1)\rho+1 \\ \cdots \\ (B-1)\rho+1 \end{bmatrix} = B + B(B-1)\rho$，$cov(x) = \sigma^2$

故上式为：$\frac{\sigma^2}{B^2}[B^2\rho + (1-\rho)B] = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$

10.3 对线性回归模型进行模拟仿真，对比 Bagging 和普通 OLS 的预测效果。

```
# y = 3 + 4x1 + 9x2 + 15x3
beta0 = 3
beta1 = 4
beta2 = 9
beta3 = 15
np.random.seed(123)
x1 = np.random.random(50)*10
x2 = np.random.random(50)*30
x3 = np.random.random(50)*20
epislon = np.random.randn(50)*np.sqrt(2)
y = np.asarray(beta0 + beta1*x1 + beta2*x2 + beta3*x3 + epislon)
X = np.asarray([np.ones(50),x1,x2,x3]).T

def ols(X,y):
    return np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)
```

```
class ols:
    def __init__(self,X,y):
        self.X = X
        self.y = y

    def fit(self):
        self.para =
            np.linalg.inv(self.X.T.dot(self.X)).dot(self.X.T).dot(self.y)

    def predict(self,X):
        # 使用MSE 作为评价指标
        y_hat = self.para[0] + X.dot(self.para[1:])
        y_true = 3 + X.dot(np.array([4,9,15]).T)
        return np.sum((y_hat-y_true)**2)

class ols_bagging:
    def __init__(self,X,y):
        self.X = X
        self.y = y

    def fit(self, n):
        # n 代表基回归器的数目
        self.para = []
        for _ in range(n):
            idx = np.random.choice(np.arange(len(self.X)),size =
                len(self.X), replace = True)
            X = self.X[idx]
            y = self.y[idx]
            self.para.append(np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y))
        self.para = np.array(self.para)

    def predict(self,X):
        y_true = 3 + X.dot(np.array([4,9,15]).T)
        y_hat =
            np.mean(np.c_[np.ones(len(X)).T,X].dot(self.para.T),axis=1)
        return np.sum((y_hat-y_true)**2)

# contruct test set
xt1 = np.random.random(50)*10
xt2 = np.random.random(50)*30
xt3 = np.random.random(50)*20
```

```
Xtest = np.asarray([xt1,xt2,xt3]).T

myols = ols(X,y)
myols.fit()
myols.para
myols.predict(Xtest)

myolsbagging = ols_bagging(X,y)
myolsbagging.fit(20)
myolsbagging.para
myolsbagging.predict(Xtest)
```

10.4 在股票涨跌预测的案例分析中，使用其他的预测指标（如 RSI、ROC 等技术指标），重新分析数据，并比较各个方法。

```
import numpy as np
import matplotlib.pyplot as plt
import talib
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# data
n = 500
n1 = 50
train = retx[-n:-n1,:]
test = retx[-n1:,:]
ret_train = train[1:,:].ravel()
ret_test = test[1:,:].ravel()
y_train = (ret_train>0).astype(int)
y_test = (ret_test>0).astype(int)
closedt = close[-(n+14):,:]

# RSI
rsi =
    np.apply_along_axis(talib.RSI,axis=0,arr=closedt,timeperiod=14)
rsi_train = rsi[-n:-n1-1,:].ravel()[:,np.newaxis]
rsi_test = rsi[-n1:-1,:].ravel()[:,np.newaxis]
clf = RandomForestClassifier(n_estimators=10,min_samples_leaf=2)
clf.fit(rsi_train,y_train)
y_pred = clf.predict(rsi_test)
print(classification_report(y_test,y_pred))
```

```
np.corrcoef([y_test,y_pred])

# 构建投资组合
holding_matrix = np.zeros((n1-1,300))
for j in range(n1-1):
    prob = clf.predict_proba(rsi[-n1+j].reshape(-1,1))[:,1]
    long_position = prob.argsort()[-10:]
    short_position = prob.argsort()[:10]
    holding_matrix[j,long_position] = .05
    holding_matrix[j,short_position] = .05
tmp_ret = np.sum(holding_matrix*test[1:,],axis=1)
portfolio_ret = np.append(0,tmp_ret)
plt.plot(np.cumprod(1+portfolio_ret))
```

10.5 配对排序学习。我们在这里描述一种将回归问题转化为分类问题的排序学习算法。假设模型为 $Y = f(X) + \epsilon$，观测数据为 $\{(y_i, \mathbf{x}_i), i = 1, \cdots, N\}$。在配对排序学习中，我们寻找 f^*，使得满足 $[f^*(\mathbf{x}_i) - f^*(\mathbf{x}_j)] \times I(y_i > y_j) > 0$ 的数据对 (i, j) 尽可能多。令 $z_{ij} = I(y_i > y_j)$，最小化:

$$\sum_{i=1}^{N} \sum_{j \neq i} \ell(z_{ij}, f^*, \mathbf{x}_i, \mathbf{x}_j)$$

其中，$\ell(z_{ij}, f^*, \mathbf{x}_i, \mathbf{x}_j) = \exp\{-z_{ij}[f^*(\mathbf{x}_i) - f^*(\mathbf{x}_j)]\}$。请编写模拟仿真程序实现配对排序学习。

答: 待补充。

11 深度学习

11.1 写出和逻辑回归等价的神经网络模型。

答: 假设样本来自两类，样本的维度为 p 维。则根据 Logistic Model，有：

$$P(Y=1|X,\beta)=\frac{1}{exp(-(\beta_0+X\beta))}=\sigma(\beta_0+X\beta)$$

因此与之等价的神经网络模型为：输入层为 p 个神经元，输出层为一个神经元，且输出层的激活函数为 sigmoid 函数。

11.2 写出卷积运算的数学表达式。

答: 假设有两个连续函数 $f(x)$、$g(x)$，则卷积运算的数学表达式为：

$$S(t)=\int f(x)\cdot g(t-x)dx$$

类似地，如果是离散函数，则卷积运算的数学表达式为：

$$S(t)=\sum_{-\infty}^{\infty}f(x)\cdot g(t-x)dx$$

11.3 把主成分分析看成自编码，写出对应的编码和解码运算。

答: 主成分分析从最小重构误差的角度来看。设观测数据为 $\{x_1,x_2,...,x_N\}$，其中 x_i 是 p 维向量，x_i 的一个 q 阶线性表示为：

$$\hat{x}_i=\mu+V\xi_i$$

最小化：

$$\sum_{i=1}^{N}\|x_i-\hat{x}_i\|_2^2=\sum_{i=1}^{N}\|x_i-\mu-V\xi_i\|_2^2$$

得到：

$$\begin{cases}\hat{\mu}=\bar{x}\\ \hat{\xi}_i=V^T(x_i-\bar{x})\end{cases}$$

其中 V 是 $\hat{\sum}=cov(X)$ 的 q 个最大特征向量。

因此，由于 ξ_i 是 q 维，其可视作编码输出，即对应 $f(x)=V^T(x-\bar{x})$，同理，解码输出为：

$$g(h)=\bar{x}+Vh$$

11.4 在 5.2.3 的案例分析中，使用其他的预测指标（如 RSI、ROC 等技术指标）重新分析数据，并调参观察神经网络的效果。

```
import numpy as np
import matplotlib.pyplot as plt
import talib
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimize import RMSprop,Adam
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

n = 500
n1 = 50
train = retx[-n:-n1,:]
test = retx[-n1:,:]
ret_train = train[1:,:].ravel()
ret_test = test[1:,:].ravel()
y_train = (ret_train>0).astype(int)
y_test = (ret_test>0).astype(int)
closedt = close[-(n+14):,:]
rsi =
    np.apply_along_axis(talib.RSI,axis=0,arr=closedt,timeperiod=14)
rsi_train = rsi[-n:-n1-1,:].ravel()[:,np.newaxis]
rsi_test = rsi[-n1:-1,:].ravel()[:,np.newaxis]

model = Sequential([
    Dense(8, input_dim=5),
    Activation('relu'),
    Dense(8),
    Activation('relu'),
    Dense(8),
    Activation('relu'),
    Dense(1),
    Activation('sigmoid'),
])
```

```
model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['binary_accuracy'])
model.fit(rsi_train,y_train,epochs=100,batch_size=200)

y_pred = model.predict(rsi_test)
print(classification_report(y_test,y_pred))
np.corrcoef([y_test,y_pred])

# 构建投资组合
holding_matrix = np.zeros((n1-1,300))
for j in range(n1-1):
    prob = model.predict_proba(rsi[-n1+j].reshape(-1,1))[:,1]
    long_position = prob.argsort()[-10:]
    short_position = prob.argsort()[:10]
    holding_matrix[j,long_position] = .05
    holding_matrix[j,short_position] = .05

tmp_ret = np.sum(holding_matrix*test[1:,],axis=1)
portfolio_ret = np.append(0,tmp_ret)
plt.plot(np.cumprod(1+portfolio_ret))
```

11.5 在 11.1.4 节的前馈神经网络中加入 Batch Normalization，推导反向传播公式。

答: 确定可学习参数，并求出 loss function 关于参数的偏导，即可使用 SGD 方法求解。

首先，加入 BN 后的模型，输入层第 i 个数据（i=1,2,...,m）c_i，线性变换后为 $x_i = Wc_i + b$，经过 BN 层后：

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}},$$

其中，

$$\begin{cases} \mu = \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \end{cases}$$

BN 层输出：

$$\hat{y}_i = \alpha\hat{x}_i + \beta$$

之后对 $\hat{y}_i$ 激活，得到 Loss 函数为：

$$Loss = \frac{1}{m}\sum_{j=1}^{m}(y_i - \theta^T\sigma(\hat{y}_i))^2$$

由此可训练参数包括：(θ,α,β,W,b)。

① $\frac{\partial L}{\partial \theta} = \frac{-2}{m}\sum_{i=1}^{m}(y_i - \theta^T\sigma(\hat{y}_i)) \cdot \sigma(\hat{y}_i)$.

② 求 $\frac{\partial L}{\partial \alpha}$ 和 $\frac{\partial L}{\partial \beta}$:

由式 $\hat{y}_i = \alpha\hat{x}_i + \beta$，可得：

$$\frac{\partial L}{\partial \alpha} = \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \alpha}$$
$$\frac{\partial L}{\partial \beta} = \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \beta}$$

而：

$$\begin{cases} \frac{\partial \hat{y}_i}{\partial \alpha} = \hat{x}_i \\ \frac{\partial \hat{y}_i}{\partial \beta} = \mathbf{1} \end{cases}$$

故只需 $\frac{\partial L}{\partial \tilde{y}_i}$。而 $\frac{\partial L}{\partial \hat{y}_i} = \frac{-2}{m}(y_i - \theta^T\sigma(\tilde{y}_i)) \cdot \theta \cdot \sigma'(\hat{y}_i)$.

③ 求 $\frac{\partial L}{\partial w_{(j)}}$ 和 $\frac{\partial L}{\partial b}$，其中 $w_{(j)}$ 是 W 的第 j 列向量。

由于 $x_i = Wc + b = \sum_{j=1}^{p} w_{(j)}c_j + b$，得到：

$$\begin{cases} \frac{\partial L}{\partial w_{(j)}} = \frac{\partial L}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_{(j)}} = \sum_{i=1}^{m}\frac{\partial L}{\partial x_i} \cdot c_j \\ \frac{\partial L}{\partial b} = \sum_{i=1}^{m}\frac{\partial L}{\partial x_i} \end{cases}$$

又有：

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial L}{\partial \mu} \cdot \frac{\partial \mu}{\partial x_i} + \frac{\partial L}{\partial \sigma^2} \cdot \frac{\partial \sigma^2}{\partial x_i}$$

第一项：

$$\frac{\partial L}{\partial \hat{x}_i} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial \hat{x}_i} = \alpha \cdot \frac{\partial L}{\partial \hat{y}_i}$$
$$\frac{\partial \hat{x}_i}{\partial x_i} = \frac{1}{\sqrt{\sigma^2 + \varepsilon}}$$

第三项：

$$\frac{\partial L}{\partial \sigma^2} = \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \sigma^2} = \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i} \cdot (-\frac{1}{2}) \cdot (x_i - \mu)(\sigma^2 + \varepsilon)^{-\frac{3}{2}}$$
$$\frac{\partial \sigma^2}{\partial x_i} = \frac{1}{m} \cdot 2(x_i - \mu)$$

第二项：

$$\begin{aligned}\frac{\partial L}{\partial \mu} &= \frac{\partial L}{\partial \sigma^2}\cdot\frac{\partial \sigma^2}{\partial \mu} + \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i}\cdot\frac{\partial \hat{x}_i}{\partial \mu} \\ &= \frac{\partial L}{\partial \sigma^2}\cdot\frac{\sum_{i=1}^{m}-2(x_i-\mu)}{m} + \sum_{i=1}^{m}\frac{\partial L}{\partial \hat{x}_i}\cdot\frac{-1}{\sqrt{\sigma^2+\varepsilon}}\end{aligned}$$

其中 $\frac{\partial L}{\partial \sigma^2}$、$\frac{\partial L}{\partial x_i}$ 已经得到。

综合三项可得出 $\frac{\partial L}{\partial x_i}$，然后将其代入③中即可。

11.6 参考 11.4.2 节，使用降噪自编码分析手写数字 3 的特征。

```
from keras.layers import Input, Dense
from keras.models import Model
from keras import regularizers
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

path = 'D:/data/zip.train.txt'
data = np.loadtxt(path)

id3 = data[:,0] == 3
data3 = data[id3,1:]
j = 330
plt.subplot(1,2,1)
plt.imshow(data3[j,:].reshape(16,16))
mean3 = np.mean(data3,axis = 0)
plt.subplot(1,2,2)
plt.imshow(mean3.reshape(16,16))

encoding_dim = 25
input_img = Input(shape=(256,))
encoded = Dense(encoding_dim, activation = 'relu')(input_img)
decoded = Dense(256, activation = 'sigmoid')(encoded)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer = 'adam', loss =
    'mean_squared_error')

def add_noise(a, percent = 0.1):
```

```
    """randomly set part of array a to 0,
    and the percentage can be set via percent"""

    id = np.random.choice(np.arange(256),size = int(256*percent),
        replace = False)
    a[id] = 0
    return a

x_train, x_test = train_test_split(data3, test_size = 0.2)
x_train_noisy = np.apply_along_axis(add_noise, axis = 1, arr =
    x_train)
x_test_noisy = np.apply_along_axis(add_noise, axis = 1, arr =
    x_test)
autoencoder.fit(x_train_noisy, x_train, epochs = 1000, batch_size
    = 32,
    shuffle = True, validation_data = (x_test_noisy, x_test),
        verbose = 2)
encoder = Model(input_img, encoded)
encoded_imgs = encoder.predict(x_train_noisy)

encoded_imgs[0]

j = 1 # 1, 2, 3...
id = encoded_imgs[:,j].argsort()
for i in range(5):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[id[-i-1]].reshape(16,16))

for i in range(5):
    plt.subplot(2, 5, 5+i+1)
    plt.imshow(x_train[id[i]].reshape(16,16))
```

11.7 在式 (11.3.25) 中，推导第 2 项 $KL\left[p_\theta(\mathbf{h}|\mathbf{x}_i)||q(\mathbf{h})\right]$ 的显式表达，并求出导数。其中，$q(\mathbf{h})$ 是标准多元正态分布，$p_\theta(\mathbf{h}|\mathbf{x}) = N\left[\mathbf{h}; \boldsymbol{\mu}_{\theta_1}(\mathbf{x}), \boldsymbol{\Sigma}_{\theta_2}(\mathbf{x})\right]$。

答: 若 $P_\theta(h|x) \sim N_k(\mu_{\theta_1}(x), \Sigma_{\theta_2}(x))$（k 值正态分布），其中 $\Sigma_{\theta_2}(x)$ 为对角阵，则各分量独立，有 $P_\theta(h|x)$=$P_\theta(h_1|x)...P_\theta(h_k|x)$。

同理，$q(h) \sim N_k(0, \mathbf{1})$，有 $q(h) = q(h_1)...q(h_k)$。

而:

$$\begin{aligned} KL(P_\theta(h|x_i)||q(h)) &= \int P_\theta(h|x_1)\cdot log\frac{P_\theta(h|x_i)}{q(h)}dh \\ &= \int P_\theta(h_1|x)...P_\theta(h_k|x)\cdot log\frac{P_\theta(h_1|x)...P_\theta(h_k|x)}{q(h_1)...q(h_k)}dh \\ &= \int P_\theta(h_1|x)...P_\theta(h_k|x)\cdot (log\frac{P_\theta(h_1|x)}{q(h_1)}+...+log\frac{P_\theta(h_k|x)}{q(h_k)})dh \\ &= \sum_{i=1}^{k}\int P_\theta(h_i|x)\cdot log\frac{P_\theta(h_i|x)}{q(h_i)}dh_i \end{aligned}$$

故只需求一元正态分布的情况即可。

由于 $P_\theta(h_i|x)\sim N(\mu_i,\sigma_i^2)$，

$$\begin{aligned} \int_{-\infty}^{\infty}P_\theta(h_i|x)log\frac{P_\theta(h_i|x)}{q(h_i)}dh_i &= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma_i}exp\{-\frac{(h_i-\mu_i)^2}{2\sigma_i^2}\}\cdot log\frac{\frac{1}{\sqrt{2\pi}\sigma_i}exp\{-\frac{(h_i-\mu_i)^2}{2\sigma_i^2}\}}{\frac{1}{\sqrt{2\pi}}exp\{-\frac{h_i^2}{2}\}}dh_i \\ &= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma_i}exp\{-\frac{(h_i-\mu_i)^2}{2\sigma_i^2}\}\cdot\left[-ln\sigma_i+\frac{h_i^2}{2}-\frac{(h_i-\mu_i)^2}{2\sigma_i^2}\right]dh_i \end{aligned}$$

此积分为三项之和，第一项为 $-ln\sigma_i$，第二项为 $\frac{1}{2}E[h_i^2]=\frac{1}{2}(\mu_i^2+\sigma_i^2)$，第三项为 $-\frac{1}{2}E[\frac{(h_i-\mu_i)^2}{\sigma_i^2}]=-\frac{1}{2}$。故整理后为：$\frac{1}{2}(\mu_i^2+\sigma_i^2-1-ln\sigma_i^2)$。

则：

$$KL(P_\theta(h_i|x)||q(h))=\sum_{i=1}^{k}\frac{1}{2}(\mu_i^2+\sigma_i^2-1-ln\sigma_i^2)$$

12 强化学习

12.1 证明由式 (12.1.15) 定义的贪婪决策函数 π'，对任意的 π 和状态 s，满足 $v_{\pi'} \geq v_{\pi}$。

答: 定义：$\forall \pi$，$\pi'(s) = \arg\max_a q_\pi(s,a)$，有：

$$\pi'(a \mid s) = \begin{cases} 1, a = \pi'(s) \\ 0, \text{其他} \end{cases}$$

所以 $\forall \pi_1$，$\forall s \in \mathcal{S}$ 有：

$$\begin{aligned} v_{\pi'}(s) &= \sum_a \pi'(a \mid s) q_{\pi'}(s,a) \\ &= \max_a q_\pi(s,a) \\ &\geqslant \max_a \ q_{\pi_1}(s,a) \\ &= \max_a q_{\pi_1}(s,a) \sum_a \pi_1(a \mid s) \\ &\geqslant \sum_a \pi_1(a \mid s) q_{\pi_1}(s,a) \\ &= v_{\pi_1}(s) \end{aligned}$$

12.2 证明由式 (12.1.18) 定义的 $\epsilon-$ 贪婪决策函数 π'，满足 $v_{\pi'} \geq v_{\pi}$。其中 π 是改进前的一个 $\epsilon-$ 贪婪决策函数。

答: 根据策略改进定理的结论, 要证明 $\pi' \geqslant \pi$，我们只需证明

$$\sum_a \pi'(a \mid s) q_\pi(s,a) \geqslant v_\pi(s), \quad s \in \mathcal{S}$$

考虑到 $\sum_a \pi'(a \mid s) q_\pi(s,a) = \frac{\varepsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\varepsilon) \max_a q_\pi(s,a)$

注意到 $1-\epsilon > 0$ 且 $1-\varepsilon = \sum_a \left(\pi(a \mid s) - \frac{\varepsilon}{|\mathcal{A}(s)|} \right)$

所以

$$\begin{aligned} (1-\varepsilon) \max_a q_\pi(s,a) &= \sum_a \left(\pi(a \mid s) - \frac{\varepsilon}{|\ \mathcal{A}(s)} \right) \max_a q_\pi(s,a) \\ &\geqslant \sum_a \left(\pi(a \mid s) - \frac{\varepsilon}{|\mathcal{A}(s)|} \right) q_\pi(s,a) \\ &= \sum_a \pi(a \mid s) q_\pi(s,a) - \frac{\varepsilon}{|\ \mathcal{A}(s)} \sum_a q_\pi(s,a) \end{aligned}$$

进而有

$$\begin{aligned}\sum_a \pi'(a \mid s) q_\pi(s,a) &= \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + (1-\varepsilon)\max_a q_\pi(s,a) \\ &\geqslant \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) + \sum_a \pi(a \mid s) q_\pi(s,a) - \frac{\varepsilon}{|\mathcal{A}(s)|}\sum_a q_\pi(s,a) \\ &= \sum_a \pi(a \mid s) q_\pi(s,a)\end{aligned}$$

12.3 描述 TD(λ) 算法。

答: TD(λ) 是历史上具有重要影响力的强化学习算法，在离线 λ 回报算法的基础上改进而来。以基于动作价值的算法为例，在离线 λ 回报算法中，对任意 n=1,2,..., 在更新 $q(S_{t-n}, A_{t-n})$ 时，时序差分目标 U_{t-nt} 的权重是 $(1-\lambda)\lambda^{n-1}$。虽然等到回合结束才能计算 U_{t-n}^{Λ}，但是在知道 (S_t, A_t) 后就能计算 $U_{t-n:t}$。所以我们在知道 (S_t, A_t) 后，就可以试图部分更新 $q(S_{t-n}, A_{t-n})$。考虑对所有的 n=1,2,... 都是如此，所以在知道 (S_t, A_t) 后就可以用 $q(S_t, A_t)$ 去更新所有的 $q(S_\tau, A_\tau)(\tau$=0,1,...,t-1)，并且更新的权重与 $\lambda^{t-\tau}$ 成正比。

据此，给定轨迹 $S_0, A_0, R_1, S_1, A_1, R_2, ...$, 可以引入资格迹 $e_t(s,a)$ 来表示第 t 步的状态动作对 (S_t, A_t) 的单步自益结果 $U_{t:t+1} = R_{t+1} + \gamma q(S_{t+1}, A_{t+1})$ 对每个状态动作对 (s,a) 需要更新的权重。

资格迹用下列递推式定义：
当 t=0 时，$e_0(s,a) = 0, s \in \mathcal{S}, a \in \mathcal{A}$;
当 t>0 时，

$$e_t(s,a) = \begin{cases} 1 + \beta\gamma\lambda e_{t-1}(s,a), & S_t = s, A_t = a \\ \gamma\lambda e_{t-1}(s,a), & \text{其他} \end{cases}$$

其中 $\beta \in [0,1]$ 是事先给定的参数，可以理解为对更新权重进行的某种强化的强度，β 常有以下取值：
$\beta = 1$ 时，这时的资格迹称为累积迹;
$\beta = 1 - \alpha$(其中 α 是学习率，这时的资格迹称为荷兰迹;
$\beta = 0$，这时的资格迹称为替换迹。
利用资格迹，可以得到 $TD(\lambda)$ 策略评估算法。在单步时序差分的基础上，加入资格迹能实现 $TD(\lambda)$ 评估动作价值的算法。资格迹也可以用于状态价值，给定轨迹 $S_0, A_0, R_1, S_1, A_1, R_2, ...$，资格迹 $e_t(s), s \in \mathcal{S}$ 来表示第 t 步的状态动作对 S_t 对的单

步自益结果 $U_{t:t+1} = R_{t+1} + \gamma v(S_{t+1})$ 对每个状态 s 需要更新的权重，其定义为：
当 t=0 时，$e_0(s) = 0, s \in \mathcal{S}$;
当 t >0 时，

$$e_t(s) = \begin{cases} 1 + \beta\gamma\lambda e_{t-1}(s), & S_t = s \\ \gamma\lambda e_{t-1}(s), & \text{其他.} \end{cases}$$

12.4 描述 Double DQN 算法。

答: Q 学习会带来最大化偏差，而双重 Q 学习却可以消除最大化偏差。基于查找表的双重 Q 学习引入了两个动作价值的估计 $q^{(0)}$ 和 $q^{(1)}$，每次更新动作价值时用其中的一个网络确定动作，用确定的动作和另外一个网络来估计回报。
对于深度 Q 学习也有同样的结论。Deepmind 于 2015 年发表论文《Deep reinforcement learning with double Q-learning》，将双重 Q 学习用于深度 Q 网络，得到了双重深度 Q 网络 (Double Deep Q Network,Double DQN)。考虑到深度 Q 网络已经有了评估网络和目标网络两个网络，所以双重深度 Q 学习在估计回报时只需要用评估网络确定动作，用目标网络确定回报的估计即可。

12.5 证明策略梯度定理 (12.3.5)。

答: 策略 $\pi(\theta)$ 满足 Bellman 期望方程，即

$$\begin{aligned} &v_{\pi(\theta)}(s) = \textstyle\sum_a \pi(a \mid s; \boldsymbol{\theta}) q_{\pi(\theta)}(s,a), && s \in \mathcal{S} \\ &q_{\pi(\theta)}(s,a) = r(s,a) + \gamma \textstyle\sum_{s'} p\left(s' \mid s, a\right) v_{\pi(\theta)}\left(s'\right), && s \in \mathcal{S}, a \in \mathcal{A}(s) \end{aligned}$$

将以上两式子对 θ 求梯度，有

$$\begin{aligned} &\nabla v_{\pi(\theta)}(s) = \textstyle\sum_a q_{\pi(\theta)}(s,a) \nabla \pi(a \mid s; \boldsymbol{\theta}) + \sum_a \pi(a \mid s; \boldsymbol{\theta}) \nabla q_{\pi(\theta)}(s,a), && s \in \mathcal{S} \\ &\nabla q_{\pi(\theta)}(s,a) = \gamma \textstyle\sum_{s'} p\left(s' \mid s, a\right) \nabla v_{\pi(\theta)}\left(s'\right), && s \in \mathcal{S}, a \in \mathcal{A}(s) \end{aligned}$$

将 $\nabla q_{\pi(\theta)}(s,a)$ 的表达式代入 $\nabla v_{\pi(\theta)}(s)$ 的表达式中，有

$$\begin{aligned} \nabla v_{\pi(\theta)}(s) &= \sum_a q_{\pi(\theta)}(s,a) \nabla \pi(a \mid s; \boldsymbol{\theta}) + \sum_a \pi(a \mid s; \boldsymbol{\theta}) \gamma \sum_{s'} p\left(s' \mid s, a\right) \nabla v_{\pi(\theta)}\left(s'\right) \\ &= \sum_a q_{\pi(\theta)}(s,a) \nabla \pi(a \mid s; \boldsymbol{\theta}) + \sum_s \Pr\left[S_{t+1} = S' \mid S_t = s; \boldsymbol{\theta}\right] \gamma \nabla v_{\pi(\theta)}\left(s'\right), \quad s \in \mathcal{S} \end{aligned}$$

在策略 $\pi(\theta)$ 下，对 S_t 求上式的期望，有

$$
\begin{aligned}
&\mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_t\right)\right] \\
&\quad=\sum_s \Pr\left[S_1=s\right] \nabla v_{\pi(\theta)}(s) \\
&\quad=\sum_s \Pr\left[S_t=s\right]\left[\sum_a q_{\pi(\theta)}(s, a) \nabla \pi(a \mid s ; \theta)+\sum_{s'} \Pr\left[S_{t+1}=s' \mid S_t=s ; \theta\right] \gamma \nabla v_{\pi(\theta)}\left(s'\right)\right] \\
&\quad=\sum_s \Pr\left[S_t=s\right] \sum_a q_{\pi(\theta)}(s, a) \nabla \pi(a \mid s ; \theta) \\
&\qquad+\sum_s \Pr\left[S_t=s\right] \sum_{s'} \Pr\left[S_{t+1}=s' \mid S_t=s ; \theta\right] \gamma \nabla v_{\pi(\theta)}\left(s'\right) \\
&\quad=\sum_s \Pr\left[S_t=s\right] \sum_\sigma q_{\pi(\theta)}(s, a) \nabla \pi(a \mid s ; \theta)+\gamma \sum_{s'} \Pr\left[S_{t+1}=s' ; \theta\right] \nabla v_{\pi(\theta)}\left(s'\right) \\
&\quad=E\left[\sum_a q_{\pi(\theta)}\left(S_t, a\right) \nabla \pi\left(a \mid S_t ; \theta\right)\right]+\gamma E\left[\nabla v_{\pi(\theta)}\left(S_{t+1}\right)\right]
\end{aligned}
$$

这样就得到了从 $E\left[\nabla v_{\pi(\theta)}(S_t)\right]$ 到 $E\left[\nabla v_{\pi(\theta)}(S_{t+1})\right]$ 的递推式。注意到最终关注的梯度值是

$$
\nabla \mathrm{E}_{\pi(\theta)}\left[G_0\right]=\nabla \mathrm{E}\left[v_{\pi(\theta)}\left(S_0\right)\right]=\mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_0\right)\right]
$$

所以有

$$
\begin{aligned}
&\nabla E_{\pi(\theta)}\left[G_0\right]=\mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_0\right)\right] \\
&\quad=\mathrm{E}\left[\sum_a q_{\pi(\theta)}\left(S_0, a\right) \nabla \pi\left(a \mid S_0 ; \theta\right)\right]+\gamma \mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_1\right)\right] \\
&\quad=\mathrm{E}\left[\sum_a q_{\pi(\theta)}\left(S_0, a\right) \nabla \pi\left(a \mid S_0 ; \theta\right)\right]+\mathrm{E}\left[\sum_a \gamma q_{\pi(\theta)}\left(S_1, a\right) \nabla \pi\left(a \mid S_1 ; \boldsymbol{\theta}\right)\right]+\gamma^2 \mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_1\right)\right] \\
&\quad=\cdots \\
&\quad=\sum_{t=0}^{+\infty} \mathrm{E}\left[\sum_a \gamma^t q_{\pi(\theta)}\left(S_t, a\right) \nabla \pi\left(a \mid S_t ; \theta\right)\right]
\end{aligned}
$$

考虑到

$$
\nabla \pi\left(a \mid S_i ; \theta\right)=\pi\left(a \mid S_t ; \theta\right) \nabla \ln \pi\left(a \mid S_t ; \theta\right)
$$

所以

$$\begin{aligned}
&\mathrm{E}\left[\sum_{a}\gamma^{t}q_{\pi(\theta)}\left(S_{t},a\right)\nabla\pi\left(a\mid S_{t};\theta\right)\right]\\
&=\mathrm{E}\left[\sum_{a}\pi\left(a\mid S_{t};\theta\right)\gamma^{t}q_{\pi(\theta)}\left(S_{t},a\right)\nabla\ln\pi\left(a\mid S_{t};\theta\right)\right]\\
&=\mathrm{E}\left[\gamma^{t}q_{\pi(\theta)}\left(S_{t},A_{i}\right)\nabla\ln\pi\left(A_{t}\mid S_{t};\theta\right)\right]
\end{aligned}$$

又由于 $q_{\pi(\theta)}\left(S_{t},A_{t}\right)=\mathrm{E}\left[G_{t}\mid S_{t},A_{t}\right]$，所以

$$\begin{aligned}
\mathrm{E}\left[\sum_{0}\gamma^{t}q_{\pi(\theta)}\left(S_{t},a\right)\nabla\pi\left(a\mid S_{t};\theta\right)\right]&=\mathrm{E}\left[\gamma^{t}q_{\pi(\theta)}\left(S_{t},A_{t}\right)\nabla\ln\pi\left(A_{t},\mid S_{t};\theta\right)\right]\\
&=\mathrm{E}\left[\gamma^{t}G_{t}\nabla\ln\pi\left(A_{t},\mid S_{t};\theta\right)\right]
\end{aligned}$$

12.6 对带基准的 REINFOECE 算法，验证 $\mathrm{E}_{s\sim d_{\pi_\theta},a\sim\pi}\left[\nabla_\theta\log\pi(a|s,\theta)b(s)\right]=0$。

答: 要证明 $\mathrm{E}_{s\sim d_{\pi_\theta},a\sim\pi}(\nabla_\theta\log\pi(a|s,\theta)b(s))=0$，只需证明

$$\mathrm{E}\left[\gamma^{t}\left(G_{t}-B\left(S_{t}\right)\right)\nabla\ln\pi\left(A_{t}\mid S_{t};\theta\right)\right]=\mathrm{E}\left[\gamma^{t}G_{t}\nabla\ln\pi(A_{t}\mid S;\theta)\right]$$

由于 B 和 a 无关，所以:

$$\sum_{a}B\left(S_{t}\right)\nabla\pi\left(a\mid S_{t};\boldsymbol{\theta}\right)=B\left(S_{t}\right)\nabla\sum_{a}\pi\left(a\mid S_{t};\theta\right)=B\left(S_{t}\right)\nabla1=0$$

进而有:

$$\begin{aligned}
\mathrm{E}\left[\gamma^{t}\left(G_{t}-B\left(S_{t}\right)\right)\nabla\ln\pi\left(A_{t}\mid S_{t};\theta\right)\right]&=\sum_{a}\gamma^{t}\left(G_{t}-B\left(S_{t}\right)\right)\nabla\pi\left(a\mid S_{t};\theta\right)\\
&=\sum_{a}\gamma^{t}G_{t}\nabla\pi\left(a\mid S_{t};\boldsymbol{\theta}\right)\\
&=\mathrm{E}\left[\gamma^{t}G_{t}\nabla\ln\pi\left(A_{t}\mid S_{t};\boldsymbol{\theta}\right)\right]
\end{aligned}$$

12.7 证明确定性策略梯度定理 (12.3.10)。

答: 状态价值和动作价值满足以下关系

$$\begin{aligned}
v_{\pi(\theta)}(s)&=q_{\pi(\theta)}(s,\pi(s;\boldsymbol{\theta})), && s\in\mathcal{S}\\
q_{\pi(\theta)}(s,\pi(s;\boldsymbol{\theta}))&=r(s,\pi(s;\boldsymbol{\theta}))+\gamma\sum_{s'}p\left(s'\mid s,\pi(s;\boldsymbol{\theta})\right)v_{\pi(\theta)}\left(s'\right), && s\in\mathcal{S}
\end{aligned}$$

以上两式对 θ 求梯度，有

$$\begin{aligned}
&\nabla v_{\pi(\theta)}(s) = \nabla q_{\pi(\theta)}(s, \pi(s;\boldsymbol{\theta})), \quad s \in \mathcal{S} \\
&\nabla q_{\pi(\theta)}\left(s, \pi(s;\boldsymbol{\theta})\right) = \left[\nabla_a r(s,a)\right]_{a=\pi(s;\theta)} \nabla\pi(s;\boldsymbol{\theta}) + \\
&\quad \gamma \sum_{s'} \left\{ \left[\nabla_a p\left(s' \mid s, a\right)\right]_{a=\pi(s;\theta)} \left[\nabla\pi(s;\boldsymbol{\theta})\right] v_{\pi(\theta)}\left(s'\right) + p\left(s' \mid s, \pi(s;\theta)\right) \nabla v_{\pi(\theta)}\left(s'\right) \right\} \\
&= \nabla\pi(s;\boldsymbol{\theta}) \left[\nabla_a r(s,a) + \gamma \sum_{s'} \nabla_a p\left(s' \mid s, a\right) v_{\pi(\theta)}\left(s'\right) \right]_{a=\pi(s;\theta)} + \gamma \sum_{s'} p\left(s' \mid s, \pi(s;\theta)\right) \nabla v_{\pi(\theta)}\left(s'\right) \\
&= \nabla\pi(s;\theta) \left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{0=\pi(s;\theta)} + \gamma \sum_{s'} p\left(s' \mid s, \pi(s;\theta)\right) \nabla v_{\pi(\theta)}\left(s'\right), s \in S
\end{aligned}$$

将 $\nabla q_{\pi(\theta)}(S, \pi(s;\theta))$ 的表达式代入到 $\nabla v_{\pi(\theta)}(s)$ 的表达式中，有

$$\nabla v_{\pi(\theta)}(s) = \nabla\pi(s;\theta) \left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)} + \gamma \sum_{s'} p\left(s' \mid s, \pi(s;\theta)\right) \nabla v_{\pi(\theta)}\left(s'\right), \quad s \in \mathcal{S}$$

对上式求关于 S_t 的期望，并考虑到 $p\left(s' \mid s, \pi(s;\theta)\right) = \Pr\left[S_{t+1} = s' \mid S_t = s; \pi(\theta)\right]$(其中 t 任取)，有

$$\begin{aligned}
&E\left[\nabla v_{\pi(\theta)}\left(S_t\right)\right] \\
&= \sum_s \Pr\left[S_t = s\right] \nabla v_{\pi(\theta)}\left(S_t\right) \\
&= \sum_s \Pr\left[S_i = s\right] \left[\nabla\pi(s;\theta) \left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)} + \gamma \sum_s p\left(s' \mid s, \pi(s;\theta)\right) \nabla v_{\pi(\theta)}\left(s'\right) \right] \\
&= \sum_s \Pr\left[S_t = s\right] \left[\nabla\pi(s;\theta) \left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)} + \gamma \sum_{s'} \Pr\left[S_{t+1} = s' \mid S_t = s; \pi(\theta)\right] \nabla v_{\pi(\theta)}\left(s'\right) \right] \\
&= \sum_s \Pr\left[S_t = s\right] \nabla\pi(s;\theta) \left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)} \\
&\quad + \gamma \sum_s \Pr\left[S_t = s\right] \sum_{s'} \Pr\left[S_{t+1} = s' \mid S_t = s; \pi(\theta)\right] \nabla v_{\pi(\theta)}\left(s'\right) \\
&= \sum_s \Pr\left[S_t = s\right] \nabla\pi(s;\theta) \left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)} + \gamma \sum_{s'} \Pr\left[S_{t+1} = s'; \pi(\theta)\right] \nabla v_{\pi(\theta)}\left(s'\right) \\
&= \mathrm{E}\left[\nabla\pi(S;\theta) \left[\nabla_a q_{\pi(\theta)}(S,a)\right]_{a=\pi(s;\theta)} \right] + \gamma \mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_{t+1}\right)\right]
\end{aligned}$$

这样就得到了从 $E\left[\nabla v_{\pi(\theta)(S_t)}\right]$ 到 $E\left[\nabla v_{\pi(\theta)}(S_{t+1})\right]$ 的递推式，注意最终关注的梯度值就是

$$\nabla E_{\pi(\theta)}[G_0] = E[\nabla v_{\pi(\theta)}(S_0)],$$

所以有

$$
\begin{aligned}
&\nabla E_{\pi(\theta)}\left[G_0\right]=\mathrm{E}\left[\nabla v_{\pi(\theta)}\left(S_0\right)\right] \\
&=\mathrm{E}\left[\nabla \pi\left(S_0;\boldsymbol{\theta}\right)\left[\nabla_a q_{\pi(\theta)}\left(S_0,a\right)\right]_{a=\pi(s_0;\theta)}\right]+\gamma E[\nabla v_{\pi(\theta)}(S_1)] \\
&=\mathrm{E}\left[\nabla \pi\left(S_0;\boldsymbol{\theta}\right)\left[\nabla_a q_{\pi(\theta)}\left(S_0,a\right)\right]_{a=\pi(S_0;\theta)}\right]+\gamma E\left[\nabla \pi(S_1;\theta)\left[\nabla_a q_{\pi(\theta)}(S_1,a)\right]_{a=\pi(S_1;\theta)}\right] \\
&\quad+\gamma^2 E\left[\nabla v_{\pi(\theta)}(S_2)\right] \\
&=\cdots \\
&=\sum_{t=0}^{+\infty}\mathrm{E}\left[\gamma^t\nabla \pi\left(S_t;\boldsymbol{\theta}\right)\left[\nabla_a q_{\pi(\theta)}\left(S_t,a\right)\right]_{a=\pi(s_t;\theta)}\right],
\end{aligned}
$$

就得到和之前梯度策略定理类似的形式。对于连续动作空间中的确定性策略，更常使用的是另外一种形式：

$$
\nabla \mathrm{E}_{\pi(\theta)}\left[G_0\right]=\mathrm{E}_{S\sim\rho_{\pi(\theta)}}\left[\nabla \pi(S;\boldsymbol{\theta})\left[\nabla_a q_{\pi(\theta)}(S,a)\right]_{a=\pi(s;\theta)}\right].
$$

其中的期望是针对折扣的状态分别

$$
\rho_\pi(s)=\int_{s_0\in S} p_{s_0}\left(s_0\right)\sum_{t=0}^{+\infty}\gamma^t\Pr\left[S_t=s\mid S_0=S_0;\boldsymbol{\theta}\right]dS_0
$$

而言的。（证明：

$$
\begin{aligned}
&\nabla E_{\pi(\theta)}\left[G_0\right]=\sum_{t=0}^{+\infty}E\left[\gamma^t\nabla \pi\left(S_t;\boldsymbol{\theta}\right)\left[\nabla_a q_{\pi(\theta)}\left(S_t,a\right)\right]_{a=\pi(s_t;\theta)}\right] \\
&=\sum_{t=0}^{+\infty}\int_S p_{S_t}(s)\gamma^t\nabla \pi(s;\boldsymbol{\theta})\left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(S_t;\theta)}\mathrm{ds} \\
&=\sum_{t=0}^{+\infty}\int_s\left(\int_{s_0} p_{S_0}\left(s_0\right)\Pr\left[S_t=s\mid S_0=s_0;\theta\right]\mathrm{d}s_0\right)\gamma^t\nabla \pi(s;\boldsymbol{\theta})\left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)}\mathrm{ds} \\
&=\int_s\left(\int_{s_0} p_{s_0}\left(s_0\right)\sum_{t=0}^{+\infty}\gamma^t\Pr\left[S_t=s\mid S_0=s_0;\theta\right]ds_0\right)\nabla \pi\left(S_t;\boldsymbol{\theta}\right)\left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)}ds \\
&=\int_s \rho_{\pi(\theta)}(s)\nabla \pi(s;\boldsymbol{\theta})\left[\nabla_a q_{\pi(\theta)}(s,a)\right]_{a=\pi(s;\theta)}\mathrm{d}s \\
&=\mathrm{E}_{\rho_{\pi(\theta)}}\left[\nabla \pi(s;\boldsymbol{\theta})\left[\nabla_a q_{\pi(\boldsymbol{\theta})}(s,a)\right]_{a=\pi(s;\theta)}\right]
\end{aligned}
$$

得证。）

12.8 对比强化学习的动作-状态和有监督学习的响应变量-解释变量，理解梯度策略定理是一种加权极大似然估计的梯度求解方法。

答: 策略梯度可以写为：

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{s\sim \mathrm{d}_{\pi_\theta}, a\sim\pi}\left[q_{\pi_\theta}(s,a)\nabla_\theta \log \pi(a \mid s,\theta)\right]$$

加权极大似然估计的对数似然函数可以写为：

$$L(\theta) = \frac{1}{n}\sum_i w_i \log f\left(y_i - x_i^\top \beta\right)$$

因而，策略梯度中的求期望对应求平均，$q_{\pi_\theta}(s,a)$ 对应权重 w_i，$\log \pi(a \mid s,\theta)$ 对应对数似然函数，所以说略梯度定理是一种加权极大似然估计的梯度求解方法。

12.9 自主编程，使用 Actor-Critic 方法对 12.2.3 章节中的案例数据进行分析并对比 DQN 的结果。

```
import talib as ta
import pandas as pd
import numpy as np
import time

from stable_baselines3 import PPO, SAC, TD3, A2C, DDPG

path_HS300 = 'D:/hot/book/data/SZ399300.TXT'


hs300 = pd.read_table(path_HS300,sep='\s+',header=None,encoding =
    'gbk')
hs300[0]=hs300[0].astype(str)
hsindex = hs300

datax = hsindex
datax.columns =
    ['date','open','high','low','close','volume','turnover']

openx = np.asarray(datax['open'],dtype=np.double)
closex = np.asarray(datax['close'],dtype=np.double)
highx = np.asarray(datax['high'],dtype=np.double)
lowx = np.asarray(datax['low'],dtype=np.double)
volumex = np.asarray(datax['volume'],dtype=np.double)
turnoverx = np.asarray(datax['turnover'],dtype=np.double)

roc1 = ta.ROC(closex,1)
```

```
roc5 = ta.ROC(closex,5)
roc10 = ta.ROC(closex,10)
rsi = ta.RSI(closex,14)
ultosc = ta.ULTOSC(highx,lowx,closex)
slowk,slowd = ta.STOCH(highx,lowx,closex)
slow_diff = slowk - slowd
adx = ta.ADX(highx,lowx,closex,14)
ppo = ta.PPO(closex)
willr = ta.WILLR(highx,lowx,closex)
cci = ta.CCI(highx,lowx,closex)
bop = ta.BOP(openx,highx,lowx,closex)
std_p20 = ta.STDDEV(closex,20)
angle = ta.LINEARREG_ANGLE(closex)
mfi = ta.MFI(highx,lowx,closex,volumex)
adosc = ta.ADOSC(highx,lowx,closex,turnoverx)

## obtian training data


start_day = '20150105'
end_day = '20150714'

st = (np.asarray(datax['date']==start_day).nonzero())[0][0]
ed = (np.asarray(datax['date']==end_day).nonzero())[0][0]

features = np.column_stack((roc1, roc5,
    roc10,rsi,ultosc,slowk,slow_diff,adx,ppo,willr,\
cci,bop,std_p20,angle,mfi,adosc))


features2 = np.float64(features[st:ed,:])

close2 = closex[st:ed]


mu1 = np.mean(features2,axis = 0)
std1 = np.std(features2,axis = 0)

Xt = (features2 - mu1)/std1
Xt[Xt>3] = 3
Xt[Xt<-3] = -3
```

```
n,p = Xt.shape



import numpy as np
import gym
from gym import spaces



class envx2(gym.Env):


def __init__(self,Xt,close2):
super(envx2,self).__init__()
self.Xt=Xt
self.Yt=close2
self.idx=0


self.action_space = spaces.Box(low=-1, high=1,
shape=(1,), dtype=np.float32)

self.observation_space = spaces.Box(low=-100, high=100,
shape=(p,), dtype=np.float32)

#def states(self):
#    return dict(type='float',shape=(16,))

#def actions(self):
#    return dict(type='int', num_values=3)

def max_episode_timesteps(self):
return super().max_episode_timesteps()

def close(self):
super().close()

def reset(self):
state = self.Xt[0,:]
```

```
self.idx=0
return state

def step(self, actions):
next_state = self.Xt[self.idx+1,:]
self.idx += 1
done = self.idx >=self.Xt.shape[0]-1
reward = (self.Yt[self.idx]-self.Yt[self.idx-1])*actions[0]
info = {}
return next_state,  reward, done, info

env = envx2(Xt,close2)

from stable_baselines3.common.env_checker import check_env
check_env(env, warn=True)


from stable_baselines3.sac.policies import MlpPolicy as sacmlp
from stable_baselines3.ppo.policies import MlpPolicy as ppomlp
from stable_baselines3.td3.policies import MlpPolicy as td3mlp
from stable_baselines3.ddpg.policies import MlpPolicy as ddpgmlp

import torch as th
policy_kwargs = dict(net_arch=dict(pi=[256, 256], qf=[256, 256]))

policy_kwargs2 = dict(activation_fn=th.nn.ReLU,
net_arch=[dict(pi=[256, 256], vf=[256, 256])])


model = SAC(sacmlp, env, policy_kwargs = policy_kwargs, verbose=1)
#model = TD3(td3mlp, env, policy_kwargs = policy_kwargs, verbose=1)
#model = PPO(ppomlp, env, policy_kwargs = policy_kwargs2,verbose=1)
#model = DDPG(ddpgmlp, env, verbose=1)
model.learn(total_timesteps=3000)

obs = env.reset()
n_steps = n
ret_seq = []
act_seq = []
for step in range(n_steps):
action, _ = model.predict(obs, deterministic=True)
```

```
obs, reward, done, info = env.step(action)
ret_seq.append(reward)
act_seq.append(action)
if done:
# Note that the VecEnv resets automatically
# when a done signal is encountered
print("Goal reached!", "reward=", reward)
break




#%matplotlib qt
import matplotlib.pyplot as plt
plt.figure()
plt.plot(np.cumsum(ret_seq),linewidth = 2)



#####################################
## out-of-sample test
#####################################
start_day = '20150715'
end_day = '20151207'


#nst = (datax['date']==start_day).nonzero()[0][0]
#ned = (datax['date']==end_day).nonzero()[0][0]
nst = np.array(datax['date']==start_day).nonzero()[0][0]
ned = np.array(datax['date']==end_day).nonzero()[0][0]



#nst = 1283
#ned = 1300


featurest = np.float64(features[nst:ned,:])

close2t = closex[nst:ned]
Xtt = (featurest - mu1)/std1
```

```
Xtt[Xtt>3] = 3
Xtt[Xtt<-3] = -3


env2 = envx2(Xtt,close2t)

obs = env2.reset()
done = False
ret_seq = []
it = 0
while done is not True:
action, _ = model.predict(obs, deterministic=True)
obs, reward, done, info = env2.step(action)
ret_seq.append(reward)
it = it + 1
if done:
# Note that the VecEnv resets automatically
# when a done signal is encountered
print("Goal reached!", "reward=", reward)
break

np.sum(ret_seq)

import matplotlib.pyplot as plt
plt.figure()
plt.plot(np.cumsum(ret_seq),linewidth = 2)
```

12.10 自主编程，使用对 12.2.3 章节中的案例程序进行修改得到 AI 股评家。

答: 在 DQN 代码中，将 Brain 最后一层输出特征数由 3 改为 1，actionCnt = 1，并在主程序中将回报改为：r = (close2[j+1] - close2[j])。其余代码不变。