1. **自己提出的问题的理解：**

**问题1：**对数似然函数的极大似然估计法和梯度下降以及拟牛顿法各自的优缺点是什么，为什么在平时不适用极大似然估计法来估计参数。

讨论后的理解：首先，最大熵估计是一种思想，即在选择模型的时候，选择熵最大的分布则是最好的模型。它并不是一种具体的求解模型的方法，而在求解模型的时候，需要求对偶函数的极大化，这种情况则可以使用极大似然估计来求得。而梯度下降法和拟牛顿法则是在求极大似然估计的时候用到的一种方法，并不是相同的一种概念。

**问题2：**95页式6.5下面的不等式是怎么推出来的？

讨论后的理解：首先，很明显H(P)应当大于0，然后，由下面的解释可以知道，在X的分布是均匀分布的时候取得最大值，此时P应当为1/n，将其代入H(P)的公式中，即可得到log|X|，所以不等式成立。

1. **别人提出的问题的理解：**

**问题1：**Zw（x）为什么等于那个式子？

讨论后的理解：因为需要求得概率值，而分子上是值为当前值的联合分布的指数，为了能够得到概率值，分母应当为所有情况的总的联合分布的指数，此时为了简化公式，可以将分母设置为Zw(X)。

**问题2：**102页，条件概率分布P（Y/X）的对数似然函数为什么是那个？

讨论后的理解：首先可以看到在累乘的左边加了一个log，这是求极大似然估计的时候的常规操作，而问题在于为什么条件概率的指数是一个先验的联合分布概率。原因很简单，因为常规的极大似然估计的指数是一个次数的整数值，而在这种情况下因为分布并不是一个离散的可以计数的值，所以并不能直接求得离散的次数，但是我们知道某个条件概率出现的概率值，而指数为概率值和次数值两种情况只是在指数上差了一个1/n的系数，这对求最大值的结果并不影响，所以可以写成书中的那种形式。

**问题3：**P9输入2页，为什么将权值向量和输入向量加以扩充，式6.3 6.4就变成了6.5 6.6的样子？

讨论后的理解：之前的w为(w1,w2,w3...wn),x为(x1,x2..xn),线性公式为wx+b,书中为了简化wx+b这个式子，可以通过将w的最后一维设置为b，将x的最后一维设置成1，即w=(w1,w2..wn,b),x=(x1,x2..xn,1),此时wx的值即为之前的wx+b的值。

1. **读书计划**

1、本周完成的内容章节：《统计学习方法》第6章

2、下周计划：《统计学习方法》第7章

1. **读书摘要及理解或伪代码的具体实现**
2. **读书摘要：**

一、逻辑斯蒂回归模型

函数中μ为位置参数，γ是形状参数，也就是μ来决定分布函数的中心对称点(μ，1/2),γ越小，分布函数越瘦，在中心附近的增长速度也就越快。

1.逻辑斯蒂回归模型学习通常采用的方法是梯度下降法和拟牛顿法。

二。最大熵模型

1.最大熵原理认为在所有可能的概率模型中，熵最大的模型是最好的模型。

2.当X服从均匀分布（即取各个值的可能性相同）时熵最大。

3.特征函数在书中的例子上直观上看是表示一个x是否对应某个类y。应该是一般用来简化那种“满足某种约束的所有实例”的情况。

4.条件熵H（Y|X）表示在已知随机变量X的条件下随机变量Y的不确定性。随机变量X给定的条件下随机变量Y的条件熵H(Y|X)，其中X也是一个随机变量，只不过是已经给定，但是需要求对于所有X的值，所以最后需要取和。

1. **代码实现**

import sklearn.datasets as datasets

import numpy as np

from sklearn.linear\_model import LogisticRegression as LR

class LogisticRegression():

    def \_\_init\_\_(self,alpha=0.01,epochs=3):

        self.W = None

        self.b = None

        self.alpha = alpha

        self.epochs = epochs

    def fit(self,X,y):

        np.random.seed(10)

        self.W = np.random.normal(size=(X.shape[1]))

        self.b = 0

        for epoch in range(self.epochs):

            w\_derivate = np.zeros\_like(self.W)

            b\_derivate = 0

            for i in range(len(y)):

                w\_derivate += (y[i] - 1/(1+np.exp(-np.dot(X[i],self.W.T)-self.b)))\*X[i]

                b\_derivate += (y[i] - 1/(1+np.exp(-np.dot(X[i],self.W.T)-self.b)))

            self.W = self.W + self.alpha\*np.mean(w\_derivate,axis=0)

            self.b = self.b + self.alpha\*np.mean(b\_derivate)

        return self

    def predict(self,X):

        p\_1 = 1/(1 + np.exp(-np.dot(X,self.W) - self.b))

        return np.where(p\_1>0.5, 1, 0)

def accuracy(pred, true):

    count = 0

    for i in range(len(pred)):

        if(pred[i] == true[i]):

            count += 1

    return count/len(pred)

def normalize(x):

    return (x - np.min(x))/(np.max(x) - np.min(x))

if \_\_name\_\_ == "\_\_main\_\_":

    X = [[1.0,2.0,3.0],

        [1.0,2.2,3.3],

        [3.2,1.5,2.6]]

    y = [1,0,1]

    X\_norm = normalize(X)

    X\_train = X\_norm[:int(len(X\_norm)\*0.8)]

    X\_test = X\_norm[int(len(X\_norm)\*0.8):]

    y\_train = y[:int(len(X\_norm)\*0.8)]

    y\_test = y[int(len(X\_norm)\*0.8):]

    lr = LogisticRegression(epochs=500,alpha=0.03)

    lr.fit(X\_train,y\_train)

    y\_pred = lr.predict(X\_test)

    acc = accuracy(y\_pred, y\_test)

    print("acc", acc)

输出：acc 1.0