1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：回归方法应用于分类有哪些难点，逻辑斯蒂回归模型是怎么解决的

讨论后的理解：

难点如下：

1.线性回归模型默认了样本之间是有顺序的 2.没有好的转换方法可以将三个及以上的因变量转化为定量数据用来进行线性回归3.线性回归可能会产生小于零或大于一的数。

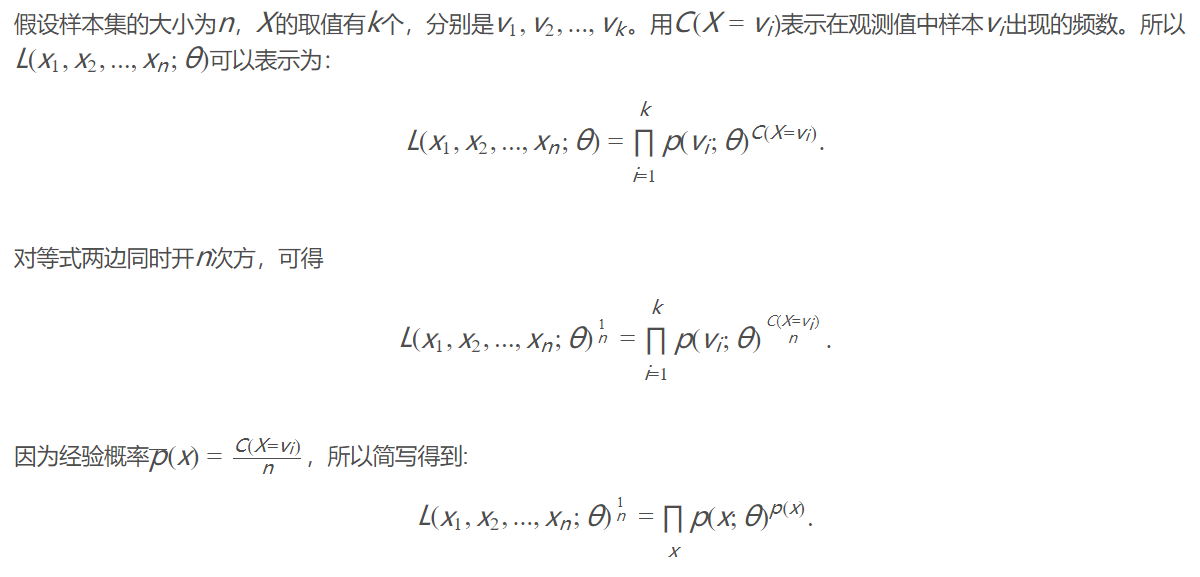
逻辑斯蒂的出发点就是利用逻辑斯蒂函数将p(X)的范围限制在[0,1]之间。

提出的问题2：逻辑斯蒂回归模型和最大熵模型有什么关系

讨论后的理解：通过合理的设计特征函数，并代入最大熵模型的概率表达式后，我们得到了逻辑斯蒂回归的条件概率表达式。因为逻辑斯蒂模型就是一个最大熵模型，它符合最大熵的思想。最大熵更加泛化，因为它未规定特征函数的具体形式。

1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. 问题3：102页，条件概率分布P（Y/X）的对数似然函数为什么是那个？

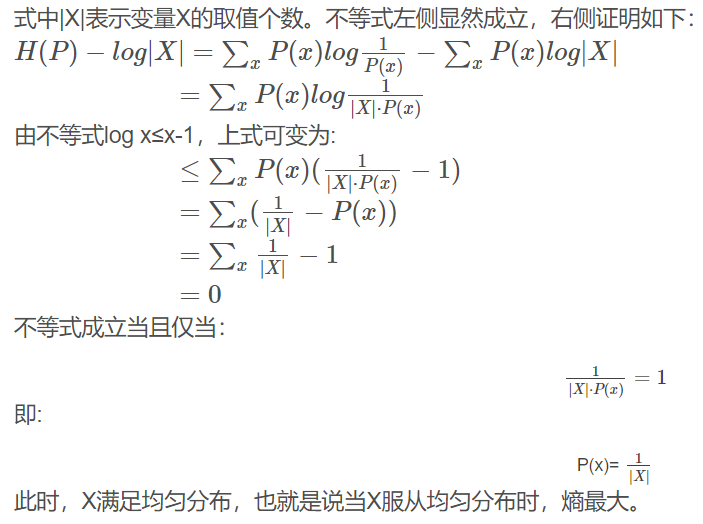
自己的理解：



显然L开n次方和L的最优值相同，因此做了一个放缩。

问题4：95页式6.5下面的不等式是怎么推出来的？

自己的理解：



问题5：P103 Z(x)为什么这样取值？

自己的理解：Z=P(X) =ΣyP(x)

三、（必填）读书计划

1、本周完成的内容章节：李航书第六章

2、下周计划：李航书附录A-C

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）

最大熵模型(maxunum entropy model)由最大熵 原理推导实现 。

最大 熵 原理 是概率模型学习的一个准则。最 大熵 原理认为，学习概率模型时， 在所有可能的概率模型(分布)中， 熵 最大的模型是最好的模型。通常用约束条 件来确定概率模型的集合，所以，最大 熵 原理也可以表述为在满足约束条件的模 型集合中选取 熵 最大的模型。均匀分布时，熵最大。

最大 熵 原理 认为要选择的概率模型首先必须满足约 束条件。在没有更多信息的情况下，那些不确定的部分都是“等可能的”。 最大 熵 原理通过 熵 的最大化来表示等可能性.“等可能”不容易操作，而 熵 则是一个 可优化的数值指标.

2.代码实现

from math import exp

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

# data

def create\_data():

iris = load\_iris()

X, y = np.array(iris.data), np.array(iris.target)

return X[:100, 0:2], y[:100]

X, y = create\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3)

class LogisticRegressionClassifier:

def \_\_init\_\_(self, max\_iter=200, learning\_rate=0.01):

self.max\_iter = max\_iter

self.learning\_rate = learning\_rate

def sigmoid(self, x):

return 1 / (1 + exp(-x))

def data\_matrix(self, X):

data\_mat = []

for d in X:

data\_mat.append([1.0, \*d])

return data\_mat

def fit(self, X, y):

# label = np.mat(y)

data\_mat = self.data\_matrix(X) # m\*n

self.weights = np.zeros((len(data\_mat[0]), 1), dtype=np.float32)

for iter\_ in range(self.max\_iter):

for i in range(len(X)):

result = self.sigmoid(np.dot(data\_mat[i], self.weights))

error = y[i] - result

self.weights += self.learning\_rate \* error \* np.transpose([data\_mat[i]]) #注意这里的[data\_mat[i]]

print('LogisticRegression Model(learning\_rate={},max\_iter={})'.format(self.learning\_rate, self.max\_iter))

# def f(self, x):

# return -(self.weights[0] + self.weights[1] \* x) / self.weights[2]

def score(self, X\_test, y\_test):

right = 0

X\_test = self.data\_matrix(X\_test)

for x, y in zip(X\_test, y\_test):

result = np.dot(x, self.weights)

if (result > 0 and y == 1) or (result < 0 and y == 0):

right += 1

return right / len(X\_test)

lr\_clf = LogisticRegressionClassifier()

lr\_clf.fit(X\_train, y\_train)

lr\_clf.score(X\_test, y\_test)

x\_ponits = np.arange(4, 8)

y\_ = -(lr\_clf.weights[1]\*x\_ponits + lr\_clf.weights[0])/lr\_clf.weights[2]

plt.plot(x\_ponits, y\_)

#lr\_clf.show\_graph()

plt.scatter(X[:50,0],X[:50,1], label='0')

plt.scatter(X[50:,0],X[50:,1], label='1')

plt.legend()

plt.show()