1. 自己提出的问题的理解：

问题1：p83式41下面说使用超平面上的任一点与边缘超平面的距离来作为d+，而不是x+到超平面的距离，为什么会这样计算？两种方式有什么本质上的区别？如果没有的话书上为什么会强调这一点？

讨论后的理解：因为最后在求边缘超平面的时候不仅仅是使用一个支持向量，而是使用全部支持向量的平均值，这样求出来的平均值不一定在原本的边缘超平面上，这样边缘超平面可能会变化，这样仅仅通过支持向量到中间超平面的距离不能代表最后的最优值，又因为两种方法求出来的结果是一样的，所以可以通过从中间超平面上的点到边缘超平面的值来求最优解。

1. 别人提出的问题的理解：

1、问题2：83的第三行，通过加入参数，我们可以调节超平面，并且不改变函数，如何理解，参数为何一定是正实数。

自己的理解：通过添加一个lamda，可以在保证中间的超平面不变的情况下（因为左右同时除以lamda可以消除lamda，所以在中间超平面上lamda的值是没有意义的），调整边缘超平面，因为边缘超平面是根据中间超平面的w来决定的，所以加入lamda会造成边缘超平面的改变。

2、问题3：软间隔svm中的损失函数为什么要选择hinge loss 这种形式，hinge loss有什么好处和局限性，有没有其他可比较的loss函数

自己的理解：一般支持向量机都会使用hinge loss这个形式的损失函数，应该是由大量的实验或者是根据支持向量机的特性来决定的，所以最优的损失函数应该是hinge loss。

3、问题4：对偶问题那里理解有点问题

自己的理解：将问题从最大化最小值转化成了最小化最大值，所以最后的结果应该是一样的，如果要深究的话就超出本书的内容，可能需要去参考一些其他的资料。

1. 读书计划

1、本周完成的内容章节：3.7-3.8

2、下周计划：第三章剩余部分

1. 读书摘要及理解或伪代码的具体实现

**(1)读书摘要：**

3.7 朴素贝叶斯分类

(1)Lidstone延续率，因为属性值有可能不是全部出现在训练集中，测试集中可能会出现新的属性值，此时ni会为0，使得所有P(Ai=ai|C=cj)=0，这个时候分类器会选择第一个判别为概率最大值的类别，不能正确的分类。对于Lidstone延续率公式，分子加常数可以避免结果为0，分母加常数用来调和因为分子改变而带来的影响。

## 3.8 支持向量机

在软边距支持向量机中之所以在目标函数中加入惩罚项是因为最小化目标函数可以在满足约束条件的情况下同时最小化惩罚项，即软边距超平面和原边距超平面尽可能相同。

**(2)代码实现：**

朴素贝叶斯的文本分类：

import pandas as pd

import numpy as np

import csv

import codecs

header = ['text', 'label']

data = [

("苹果","1"),

("桃子","1"),

("西瓜","1"),

("火龙果","1"),

("芒果","1"),

("香蕉","1"),

("电路板","0"),

("电脑","0"),

("机箱","0"),

("主机","0"),

("显示器","0"),

("适配器","0"),

("充电器","0"),

("手机","0"),

]

with open('train.csv', 'w', encoding='utf-8') as f:

writer = csv.writer(f)

writer.writerow(header)

writer.writerows(data)

def get\_data(path):

dataset = pd.read\_csv(path)

dataset = np.array(dataset)

train = dataset[:,0:dataset.shape[1]-1]

labels = dataset[:,-1]

return train,labels

train,labels = get\_data("train.csv")

class NaiveBayes(object):

def \_\_init\_\_(self,lamda,train,labels):

self.label\_map = {1:"水果",0:"机器"}

self.lamda = lamda

self.train = train

self.labels = set(labels)

#label\_prob\_pre : 每个类别的先验概率，即Pr(cj)

labels = list(self.labels)

label\_set = set(labels)

self.label\_prob\_pre = {}

for label in label\_set:

self.label\_prob\_pre[label] = labels.count(label)/float(len(labels))

#probs\_wc:Pr(w|c)

word\_list = [word for sen in self.train for word in sen[0]]

self.words\_set = set(word\_list)

self.probs\_wc = {}

for c in labels:

for word in self.words\_set:

self.probs\_wc[(word,c)]=self.get\_prob\_wc(word,c)

def get\_prob\_wc(self,word,c):

#求Pr(w|c)

num\_word\_c = 0

num\_c = 0

v = len(self.words\_set)

for i,data in enumerate(train):

label = labels[i]

if label == c:

num\_c += len(data[0])

num\_word\_c += data[0].count(word)

# print(num\_c)

return (num\_word\_c+self.lamda)/float(self.lamda\*v+num\_c)

#def get\_multi\_wc(self):

def classify(self,d):

mul\_dc\_sum = 0 #分母上对每个c，求每个word的乘积

for label in self.labels:

mul\_dc\_tmp = 1

for w in d:

mul\_dc\_tmp \*=self.probs\_wc[(w,label)]

mul\_dc\_sum +=mul\_dc\_tmp \* self.label\_prob\_pre[label]

results = []

for label in self.labels:

mul\_dc = 1

for w in d:

mul\_dc \*= self.probs\_wc[(w,label)]

results.append(self.label\_prob\_pre[label] \* mul\_dc)

print(results)

res\_c = np.argmax(results)

return self.label\_map[res\_c]

nb = NaiveBayes(0.5,train,labels)

print(nb.classify("电器"))

print(nb.classify("香瓜"))

运行结果：

[0.0057988165680473375, 0.00019223375624759708]

机器

[0.0001183431952662722, 0.0017301038062283738]

水果