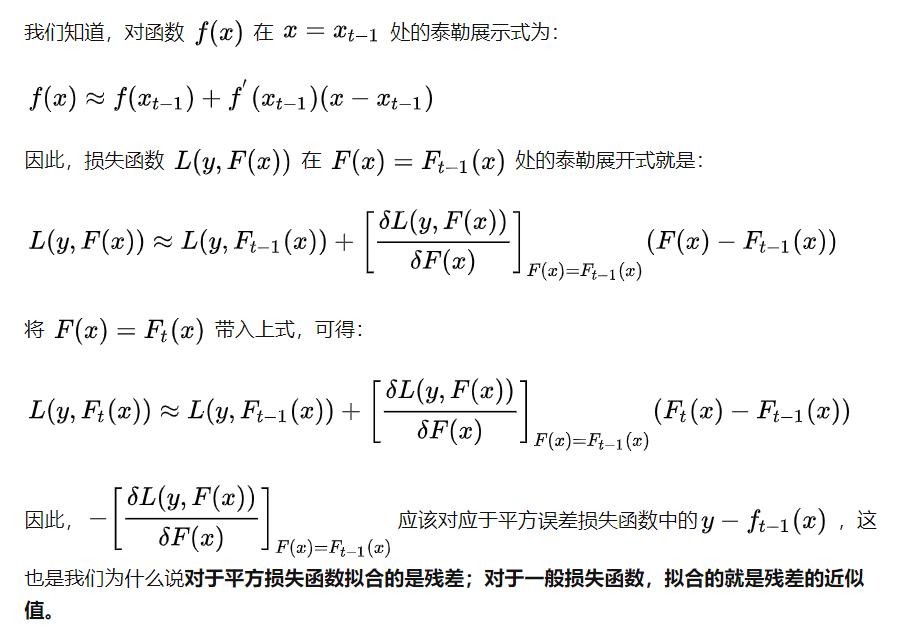
1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：170页为什么提升树对指数损失函数和平方损失函数优化简单，损失函数是根据什么选择？

讨论后的理解：

指数损失函数感觉对应的是Adaboost算法而不是提升树算法，基函数为决策树桩的boost算法（就是提升树）使用了残差的概念来优化提升效果，而平方损失函数的负梯度正好是残差或残差的倍数。优化损失函数与正则项，其实代表的是对参数的极大似然或者极大后验估计，不同的损失函数和正则项，反映的我们对参数先验分布和似然函数的不同假设。

提出的问题2：对于一般损失函数而言，为什么可以利用损失函数的负梯度在当前模型的值作为梯度提升算法中残差的近似值呢？

讨论后的理解：



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：P159 例题中 如何确定阈值v使得分类误差率最低？

自己的理解：

手动对所有的数据依次进行划分，每一次划分都计算误差率的大小，最后选择误差率最小的划分结果作为阈值V。

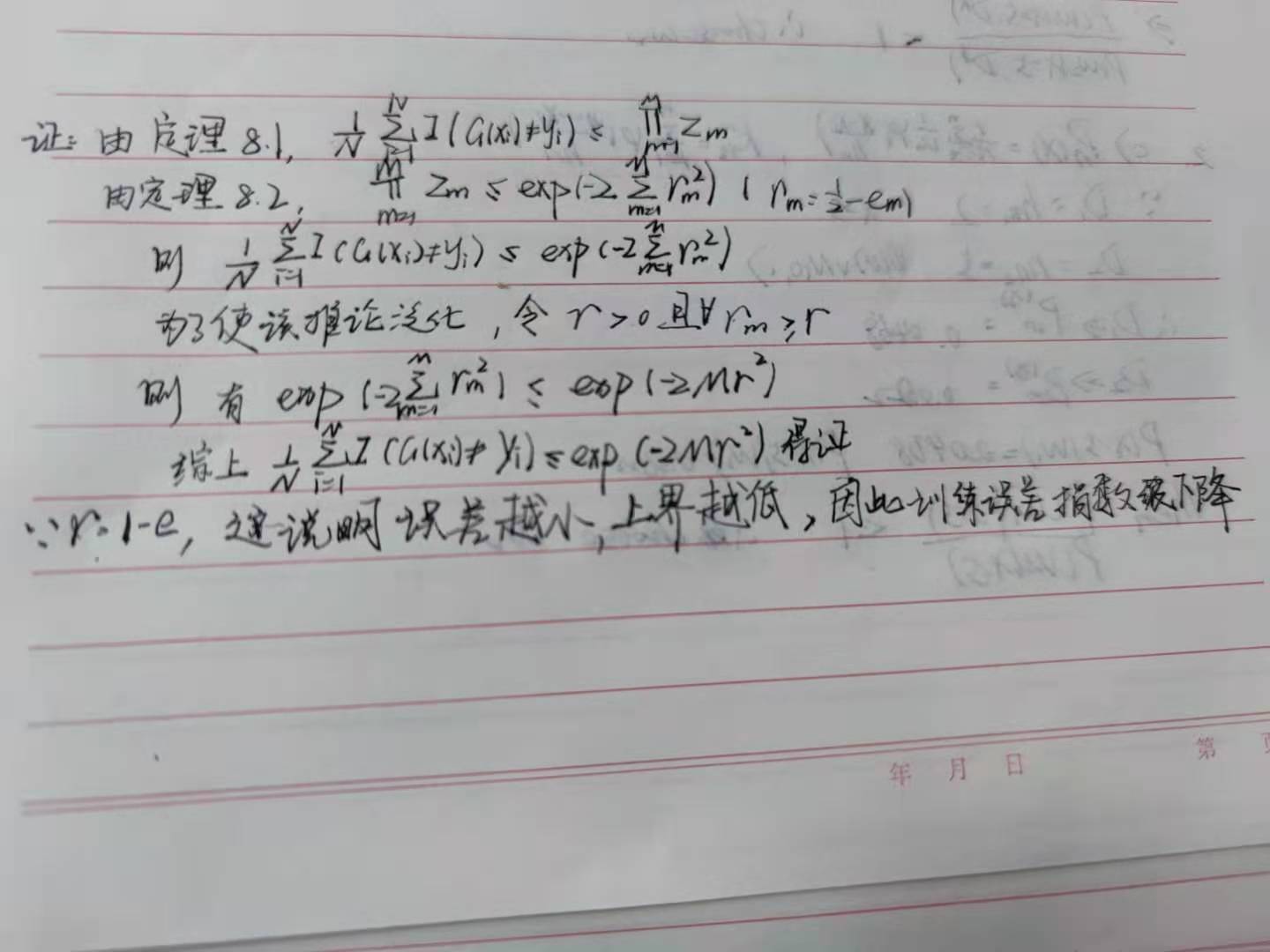
问题4：梯度提升算法（2）（b）为什么要对r拟合一个回归树？

自己的理解：

书中说，对于一般的损失函数，r（即负梯度）近似地等于回归问题中的残差，所以对于r，拟合一个回归树就可以达到优化模型的效果。对于常数c，因为对于每一个回归树，都有最优的c，可以使得损失函数最小，所以可以使用求让损失函数最小的方法来求得当前回归树的常数c.

问题5：推论8.1

自己的理解：



三、（必填）读书计划

1、本周完成的内容章节：李航书第8章

2、下周计划：李航书第9章

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）

**简述：提升方法（boosting）是一种常用的统计学习方法，应用广泛且有效。在分类问题中，它通过改变训练样本的权重，学习多个分类器，并将这些分类器进行线性组合，提高分类的性能。**

**本章**

**（1）介绍boosting方法的思路和代表性的boosting算法AdaBoost**

**（2）通过训练误差分析探讨AdaBoost为什么能提高学习精度**

**（3）从前向分布加法模型的角度解释AdaBoost**

**（4）最后叙述boosting方法更具体的实例——boosting tree（提升树）**

**boosting基本思路：boosting基于这样一种思想：对于一个复杂任务来说，将多个专家的判断进行适当的综合所得出的判断，要比其中任何一个专家单独的判断好。实际上就是“三个臭皮匠，顶个诸葛亮”的道理。**

**强可学习：在概率近似正确（probably approximately correct，PAC）学习的框架中，一个概念，如果存在一个多项式的学习算法能够学习它，并且正确率很高，那么称这个概念是强可学习的。**

**弱可学习：一个概念，如果存在一个多项式的学习算法能够学习它，学习的正确率仅比随机猜测略好，那么称为弱可学习的。**

**强可学习和弱可学习：Schapire证明了强可学习与弱可学习是等价的，也就是说，在PAC学习的框架下，一个概念是强可学习的充要条件是这个概念是弱可学习的。**

**从弱学习到强学习：可将“弱学习”提升为“强学习”，弱学习算法通常比强学习算法容易得多。具体如何实施提升，便称为开发提升方法时要解决的问题。有很多提升算法被提出，最具代表性的就是AdaBoost。**

**提升方法就是从弱学习算法出发，反复学习，得到一系列弱分类器（又称为基分类器），然后组合这些弱分类器，构成一个强分类器。大多数的提升方法都是改变训练数据的概率分布（训练数据的权重分布），针对不同的训练数据分布，调用弱学习算法学习一系列弱分类器。这里就有两个问题。**

**（1）在每一轮如何改变训练数据的权值或概率分布**

**（2）如何将弱分类器组合成一个强分类器**

2.代码实现

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

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

import matplotlib.pyplot as plt

# data

def create\_data():

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names)

df['label'] = iris.target

df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']

data = np.array(df.iloc[:100, [0, 1, -1]])

for i in range(len(data)):

if data[i,-1] == 0:

data[i,-1] = -1

# print(data)

return data[:,:2], data[:,-1]

X, y = create\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

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

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

plt.legend()

plt.show()

class AdaBoost:

def \_\_init\_\_(self, n\_estimators=50, learning\_rate=1.0):

self.clf\_num = n\_estimators

self.learning\_rate = learning\_rate

def init\_args(self, datasets, labels):

self.X = datasets

self.Y = labels

self.M, self.N = datasets.shape

# 弱分类器数目和集合

self.clf\_sets = []

# 初始化weights

self.weights = [1.0 / self.M] \* self.M

# G(x)系数 alpha

self.alpha = []

def \_G(self, features, labels, weights):

m = len(features)

error = 100000.0 # 无穷大

best\_v = 0.0

# 单维features

features\_min = min(features)

features\_max = max(features)

n\_step = (features\_max - features\_min + self.learning\_rate) // self.learning\_rate

# print('n\_step:{}'.format(n\_step))

direct, compare\_array = None, None

for i in range(1, int(n\_step)):

v = features\_min + self.learning\_rate \* i

if v not in features:

# 误分类计算

compare\_array\_positive = np.array([1 if features[k] > v else -1 for k in range(m)])

weight\_error\_positive = sum([weights[k] for k in range(m) if compare\_array\_positive[k] != labels[k]])

compare\_array\_nagetive = np.array([-1 if features[k] > v else 1 for k in range(m)])

weight\_error\_nagetive = sum([weights[k] for k in range(m) if compare\_array\_nagetive[k] != labels[k]])

if weight\_error\_positive < weight\_error\_nagetive:

weight\_error = weight\_error\_positive

\_compare\_array = compare\_array\_positive

direct = 'positive'

else:

weight\_error = weight\_error\_nagetive

\_compare\_array = compare\_array\_nagetive

direct = 'nagetive'

# print('v:{} error:{}'.format(v, weight\_error))

if weight\_error < error:

error = weight\_error

compare\_array = \_compare\_array

best\_v = v

return best\_v, direct, error, compare\_array

# 计算alpha

def \_alpha(self, error):

return 0.5 \* np.log((1 - error) / error)

# 规范化因子

def \_Z(self, weights, a, clf):

return sum([weights[i] \* np.exp(-1 \* a \* self.Y[i] \* clf[i]) for i in range(self.M)])

# 权值更新

def \_w(self, a, clf, Z):

for i in range(self.M):

self.weights[i] = self.weights[i] \* np.exp(-1 \* a \* self.Y[i] \* clf[i]) / Z

# G(x)的线性组合

def \_f(self, alpha, clf\_sets):

pass

def G(self, x, v, direct):

if direct == 'positive':

return 1 if x > v else -1

else:

return -1 if x > v else 1

def fit(self, X, y):

self.init\_args(X, y)

for epoch in range(self.clf\_num):

best\_clf\_error, best\_v, clf\_result = 100000, None, None

# 根据特征维度, 选择误差最小的

for j in range(self.N):

features = self.X[:, j]

# 分类阈值，分类误差，分类结果

v, direct, error, compare\_array = self.\_G(features, self.Y, self.weights)

if error < best\_clf\_error:

best\_clf\_error = error

best\_v = v

final\_direct = direct

clf\_result = compare\_array

axis = j

# print('epoch:{}/{} feature:{} error:{} v:{}'.format(epoch, self.clf\_num, j, error, best\_v))

if best\_clf\_error == 0:

break

# 计算G(x)系数a

a = self.\_alpha(best\_clf\_error)

self.alpha.append(a)

# 记录分类器

self.clf\_sets.append((axis, best\_v, final\_direct))

# 规范化因子

Z = self.\_Z(self.weights, a, clf\_result)

# 权值更新

self.\_w(a, clf\_result, Z)

# print('classifier:{}/{} error:{:.3f} v:{} direct:{} a:{:.5f}'.format(epoch+1, self.clf\_num, error, best\_v, final\_direct, a))

# print('weight:{}'.format(self.weights))

# print('\n')

def predict(self, feature):

result = 0.0

for i in range(len(self.clf\_sets)):

axis, clf\_v, direct = self.clf\_sets[i]

f\_input = feature[axis]

result += self.alpha[i] \* self.G(f\_input, clf\_v, direct)

# sign

return 1 if result > 0 else -1

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

right\_count = 0

for i in range(len(X\_test)):

feature = X\_test[i]

if self.predict(feature) == y\_test[i]:

right\_count += 1

return right\_count / len(X\_test)

X = np.arange(10).reshape(10, 1)

y = np.array([1, 1, 1, -1, -1, -1, 1, 1, 1, -1])

clf = AdaBoost(n\_estimators=3, learning\_rate=0.5)

clf.fit(X, y)

X, y = create\_data()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33)

clf = AdaBoost(n\_estimators=10, learning\_rate=0.2)

clf.fit(X\_train, y\_train)

print(clf.score(X\_test, y\_test))