1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：knn每个样本都要遍历一遍数据集，这样算法时间复杂度很高。如何优化knn算法？

讨论后的理解：对于N个样本的训练集，每做一次分类，时间复杂度为O(N),为了降低时间复杂度，有学者提出KD树算法，核心思想是将训练样本组织成树的形式再做搜索，这样时间复杂度降为O（logn）,具体做法是每次迭代选取一个维度将一棵树（或子树）划分成两科棵子树。

1. 提出的问题2：knn的k个近邻其实还是有相对远近的，在决策时应该如何进一步优化

讨论后的理解：一方面要根据实际情况选择合适的距离公式，因为不同距离公式可能对应不同的k近邻。另一方面主要是靠权重来优化k近邻的决策，比如近邻点的权重是距离的倒数。

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：书中的两种集成分类器方法中每个基本分类器都是使用的相同的分类算法，如果使用不同的分类算法有没有可能实验结合每种算法各自的优点的可能？

自己的理解：bagging和boosting算法都是同质学习算法，将相同的算法作为其中的弱学习器。Stacking是一种异质算法，通过并行的执行不同的弱学习器并赋予特定的权重。

1. 问题4 ：Bagging为什么会对不稳定学习算法的性能显著提高，而对于稳定的的分类器反而会降低准确率？

自己的理解：bagging通过多次试验决定最终结果，如果算法不稳定的话可以看做偶尔错，这样bagging有能力纠错。如果某个算法稳定且不正确的话，可以看做经常错，bagging将作出错误的决定

5、 问题5; knn在样本不平衡时会有缺陷，例如有一类样本容量很大，其他类样本容量小，导致大容量的样本在k个邻居中占多数。有无改进的办法？

自己的理解：对于数据不平衡问题，一般的做法是通过升采样（降采样）来调整。除此之外，可以将TF-IDF（Term Frequency-Inverse Document Frequency, 词频-逆文件频率）应用到调整权重上。换句话说，权重随着它在样本中出现的次数成正比增加，但同时会随着它在总体中出现的频率成反比下降。

1. （必填）读书计划

1、本周完成的内容章节：如3.9-3.10

2、下周计划：2.1-2.5

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）

3.9 **KNN**

3.9.1 **思路**：如果一个样本在特征空间中的k个最邻近的样本中的大多数属于某一个类别，则该样本也划分为这个类别。KNN算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

3.9.2 **流程**：

1）计算测试数据与各个训练数据之间的距离；

　　2）按照距离的递增关系进行排序；

　　3）选取距离最小的K个点；

　　4）确定前K个点所在类别的出现频率；

　　5）返回前K个点中出现频率最高的类别作为测试数据的预测分类

2、伪代码的具体实现(选做)

import numpy as np

import operator

from os import listdir

from sklearn.neighbors import KNeighborsClassifier as kNN

"""

Parameters:

filename - 文件名

Returns:

returnVect - 返回的二进制图像的1x1024向量

"""

# 函数说明:将32x32的二进制图像转换为1x1024向量。

def img2vector(filename):

#创建1x1024零向量

returnVect = np.zeros((1, 1024))

#打开文件

fr = open(filename)

#按行读取

for i in range(32):

#读一行数据

lineStr = fr.readline()

#每一行的前32个元素依次添加到returnVect中

for j in range(32):

returnVect[0, 32\*i+j] = int(lineStr[j])

#返回转换后的1x1024向量

return returnVect

# 函数说明:手写数字分类测试

def handwritingClassTest():

#测试集的Labels

hwLabels = []

#返回trainingDigits目录下的文件名

trainingFileList = listdir('trainingDigits')

#返回文件夹下文件的个数

m = len(trainingFileList)

#初始化训练的Mat矩阵,测试集

trainingMat = np.zeros((m, 1024))

#从文件名中解析出训练集的类别

for i in range(m):

#获得文件的名字

fileNameStr = trainingFileList[i]

#获得分类的数字

classNumber = int(fileNameStr.split('\_')[0])

#将获得的类别添加到hwLabels中

hwLabels.append(classNumber)

#将每一个文件的1x1024数据存储到trainingMat矩阵中

trainingMat[i,:] = img2vector('trainingDigits/%s' % (fileNameStr))

#构建kNN分类器

neigh = kNN(n\_neighbors = 3, algorithm = 'auto')

#拟合模型, trainingMat为测试矩阵,hwLabels为对应的标签

neigh.fit(trainingMat, hwLabels)

#返回testDigits目录下的文件列表

testFileList = listdir('testDigits')

#错误检测计数

errorCount = 0.0

#测试数据的数量

mTest = len(testFileList)

#从文件中解析出测试集的类别并进行分类测试

for i in range(mTest):

#获得文件的名字

fileNameStr = testFileList[i]

#获得分类的数字

classNumber = int(fileNameStr.split('\_')[0])

#获得测试集的1x1024向量,用于训练

vectorUnderTest = img2vector('testDigits/%s' % (fileNameStr))

#获得预测结果

# classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)

classifierResult = neigh.predict(vectorUnderTest)

print("分类返回结果为%d\t真实结果为%d" % (classifierResult, classNumber))

if(classifierResult != classNumber):

errorCount += 1.0

print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount/mTest \* 100))

if \_\_name\_\_ == '\_\_main\_\_':

handwritingClassTest()