1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：为什么平衡时的kd树搜索效率未必是最优的

讨论后的理解： 平衡时的kd树可以保证平均时间复杂度最优，划分空间也比较均匀，但是单次搜索效率不一定是最优的。

提出的问题2：为什么选择数据的中位数作为切分点，有没有更针对性的策略

讨论后的理解：中位数是数据最中间的点或者它的附近，可以保证划分的空间是均匀的，构造的kd树是平衡的。但是中位数需要排序才能获得，会产生额外的时间开销。在数据比较均匀的情况下选择平均数会减小时间复杂度。

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：构造kd树相当于不断用垂直与坐标轴的超平面将k维空间切分，构成一系列的k维超矩形区域，如何理解

自己的理解：这里说的"坐标轴"就是实例特征向量的k维特征，也就是有k个坐标轴；对于一个二维空间(即一个平面)，超平面就是一条直线；在二维空间中，书中的定义就等同于，用一条垂直于x轴或y轴的直线将这个平面切分，构成两个二维矩形平面(对应多维空间，就叫超矩形区域)。然后一直重复这个操作，直到最终的子区域内没有实例为止。

问题4：使用Kd树最近邻预测时，矩形和超球面容易相交，这时可能需要对一些多余的点进行搜索，有没有其他的办法

自己的理解：

1)先构造一个超球体，此超球体是可以包含所有样本的最小球体；

2)从球中选择一个离球的中心点最远的点，然后选择第二个点离第一个点最远，将球中所有的点分配到离这两个聚类中心最近的一个上，然后计算每个聚类的中心，以及聚类能够包含它所有数据点所需的最小半径。这样得到了两个子超球体，和KD树里面的左右字数对应。

3)对于这两个超球体，递归执行上一步骤，最终得到一个球树。

问题5：书上给的kd树例子是最近邻，那么k近邻要怎么实现？如何确保最后能够找到k个最近邻？

自己的理解： k近邻在搜索最近邻的基础上，忽略之前找到的最近邻实例，重新选择最近邻，重复k次，得到目标点的k个最近邻。

三、（必填）读书计划

1、本周完成的内容章节：李航书第三章

2、下周计划：李航书第四章

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）

KNN最简单的实现方法是线性扫描，这是一种暴力实现方法，然而当训练集很大时，搜索k个最近邻居的计算非常耗时。在实现kNN时，主要考虑的问题就是如何对训练数据进行快速k近邻搜索。这样的方法有kd树和球树。kd树使用了特殊的结构存储训练数据，减少计算目标值与训练实例的距离的次数。kd树算法包括两个主要步骤：1）构造kd树；2）搜索最近邻；

2.代码实现

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

from collections import Counter

# 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]])

plt.figure(figsize=(15, 8))

plt.subplot(121)

plt.scatter(df[:50]['sepal length'], df[:50]['sepal width'], label='0')

plt.scatter(df[50:100]['sepal length'], df[50:100]['sepal width'], label='1')

plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend()

plt.title('original data')

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

X, y = data[:,:-1], data[:,-1]

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

class KNN:

def \_\_init\_\_(self, X\_train, y\_train, n\_neighbors=3, p=2):

"""

parameter: n\_neighbors 临近点个数

parameter: p 距离度量

"""

self.n = n\_neighbors

self.p = p

self.X\_train = X\_train

self.y\_train = y\_train

def predict(self, X):

# 取出n个点

knn\_list = []

for i in range(self.n):

dist = np.linalg.norm(X - self.X\_train[i], ord=self.p)

knn\_list.append((dist, self.y\_train[i]))

for i in range(self.n, len(self.X\_train)):

max\_index = knn\_list.index(max(knn\_list, key=lambda x: x[0])) #这里的knn\_list相当于是一个二维数组，所以lambda中取x[0]

dist = np.linalg.norm(X - self.X\_train[i], ord=self.p)

if knn\_list[max\_index][0] > dist:

knn\_list[max\_index] = (dist, self.y\_train[i]) #注意这里的括号

# 统计

knn = [k[-1] for k in knn\_list]

count\_pairs = Counter(knn) #count\_pairs为键值对的形式

max\_count = sorted(count\_pairs.items(), key=lambda x: x[1])[-1][0] #这里的lambda变量取得是x[1]

return max\_count

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

right\_count = 0

n = 10

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

label = self.predict(X)

if label == y:

right\_count += 1

return right\_count / len(X\_test)

clf = KNN(X\_train, y\_train)

print('Test\_score: {}'.format(clf.score(X\_test, y\_test)))

test\_point = [6.0, 3.0]

print('Test Point: {}'.format(clf.predict(test\_point)))

plt.subplot(122)

plt.scatter(df[:50]['sepal length'], df[:50]['sepal width'], label='0')

plt.scatter(df[50:100]['sepal length'], df[50:100]['sepal width'], label='1')

plt.plot(test\_point[0], test\_point[1], 'bo', label='test\_point')

plt.xlabel('sepal length')

plt.ylabel('sepal width')

plt.legend()

plt.title('classification by KNN(n=3, p=2)')

plt.subplots\_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,

wspace=0.35) #调整子图间距

plt.savefig("demo.jpg")

plt.show()