模式识别第二次上机大作业

**人工智能32班 20009201208 张佳凡**

### 一、问题重述

**分析K近邻算法的错误率**

**推荐数据集**

数据： UCI数据库1中sonar和Iris数据；

自然图像：Mnist手写体数据集2 、CIFAR数据集3；

遥感图像：UCM数据集4 ，NWPU数据集5 。

**作业形式：程序+大报告+上机课演示**

1. http://archive.ics.uci.edu/ml/index.php

2. http://yann.lecun.com/exdb/mnist/

3. https://www.cs.toronto.edu/~kriz/cifar.html

4. http://vision.ucmerced.edu/datasets/landuse.html

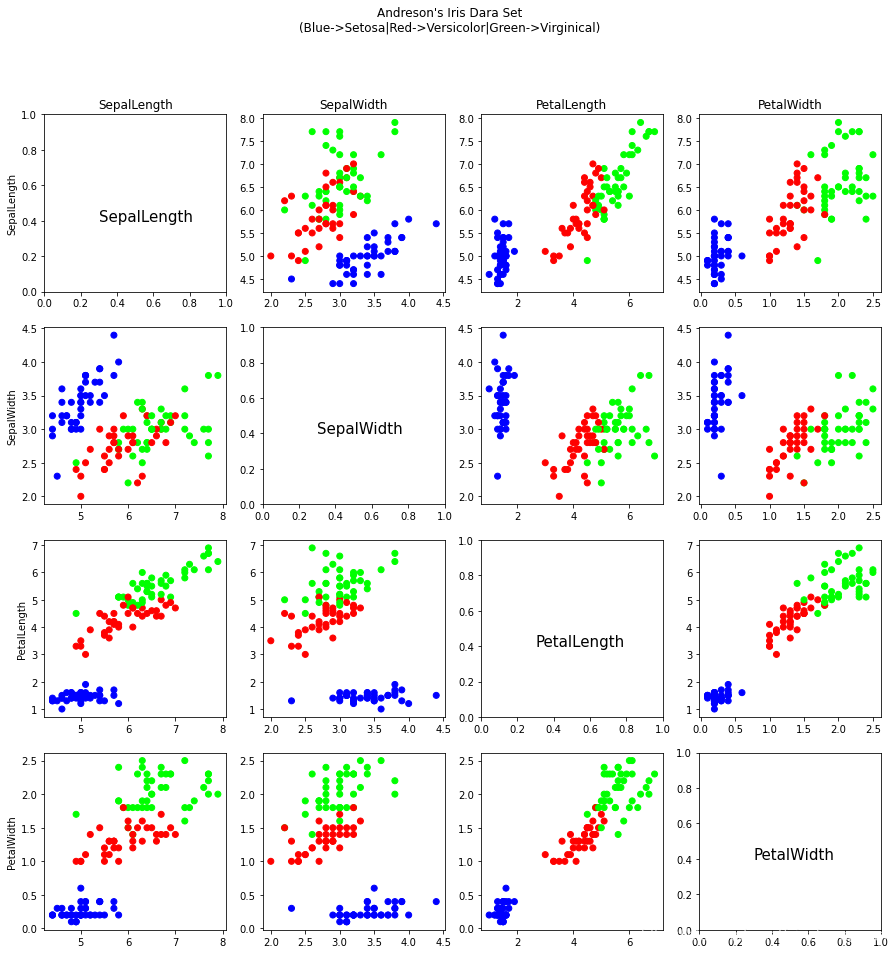
5. http://www.escience.cn/people/JunweiHan/NWPU-RESISC45.html

### 数据集介绍

在本次上机实验中，我选择了Iris鸢尾花数据集和Mnist手写字体识别数据集来验证验证KNN最近邻方法的错误率。

**Iris鸢尾花数据集**

Iris数据集以鸢尾花的特征作为数据来源，数据集包含150个数据集，有4维，分为3 类（setosa、versicolour、virginica），每类50个数据，每个数据包含4个属性，花萼长度、宽度和花瓣长度、宽度。对数据进行统计学处理及数据可视化，分别研究二维数据可视化散点图如下：

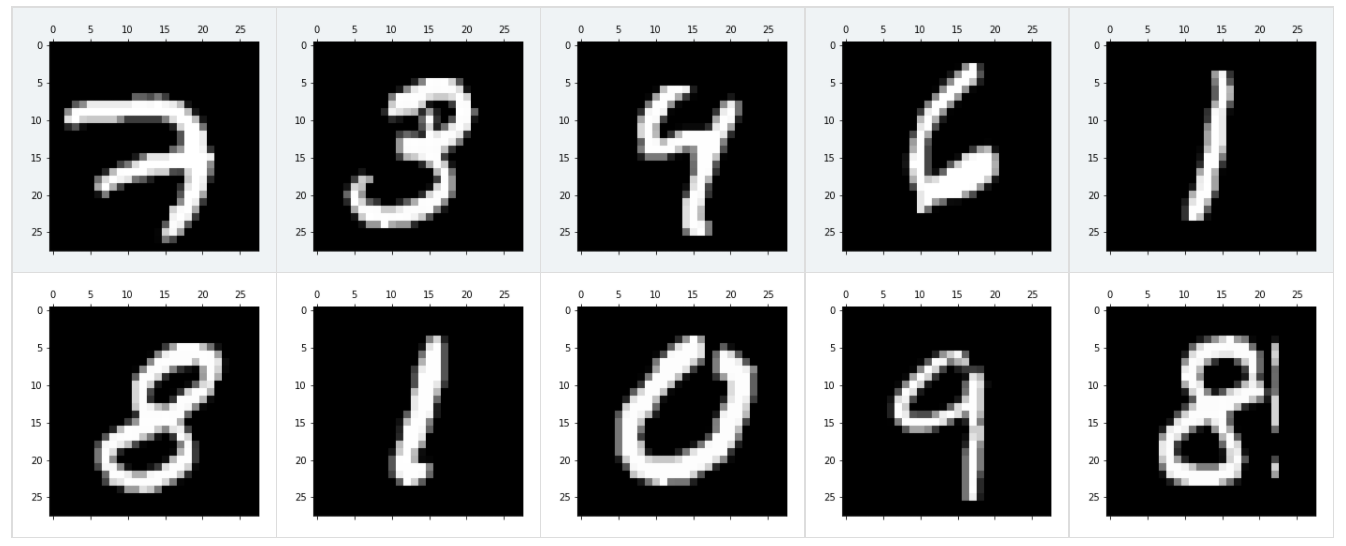


**MNIST数据集**

MNIST数据集来自美国国家标准与技术研究所, National Institute of Standards and Technology (NIST)。训练集（training set）由来自250个不同人手写的数字构成，其中50%是高中学生，50%来自人口普查局（the Census Bureau）的工作人员。测试集（test set）也是同样比例的手写数字数据，但保证了测试集和训练集的作者集不相交。

MNIST数据集一共有7万张图片，其中6万张是训练集，1万张是测试集。每张图片是28 × 28 28\times 2828×28的0 − 9 0-90−9的手写数字图片组成。每个图片是黑底白字的形式，黑底用0表示，白字用0-1之间的浮点数表示，越接近1，颜色越白。

选取Mnist数据集前十项数据展示如下：



### 实验原理

KNN算法的核心思想是，如果一个样本在特征空间中的K个最相邻的样本中的大多数属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。该方法在确定分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。KNN方法在类别决策时，只与极少量的相邻样本有关。由于KNN方法主要靠周围有限的邻近的样本，而不是靠判别类域的方法来确定所属类别的，因此对于类域的交叉或重叠较多的待分样本集来说，KNN方法较其他方法更为适合。

总体来说，KNN分类算法包括以下4个步骤： 

①准备数据，对数据进行预处理 。

②计算测试样本点（也就是待分类点）到其他每个样本点的距离 。

我们选取欧氏距离作为距离度量：、

③对每个距离进行排序，然后选择出距离最小的K个点  。

④对K个点所属的类别进行比较，根据[少数服从多数](https://baike.baidu.com/item/%E5%B0%91%E6%95%B0%E6%9C%8D%E4%BB%8E%E5%A4%9A%E6%95%B0/10350157?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%82%BB%E8%BF%91%E7%AE%97%E6%B3%95/_blank)的原则，将测试样本点归入在K个点中占比最高的那一类

### 四、实验结果及分析

**1. Iris鸢尾花数据集**

使用Iris数据集，进行KNN算法运算，我们改变K的取值来研究不同K值下KNN在Iris数据集上的准确率，选取K值为1~10，得到分类准确率如下**：**

K: 1 accuracy: 0.9629629629629628

K: 2 accuracy: 0.9629629629629628

K: 3 accuracy: 0.9629629629629628

K: 4 accuracy: 0.9629629629629628

K: 5 accuracy: 0.9629629629629628

K: 6 accuracy: 0.9629629629629628

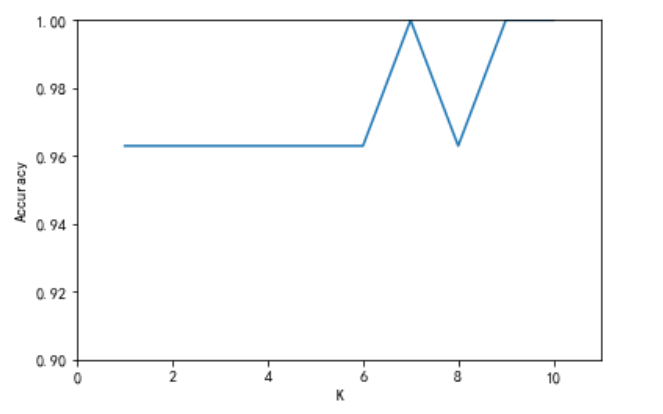
K: 7 accuracy: 0.9999999999999998

K: 8 accuracy: 0.9629629629629628

K: 9 accuracy: 0.9999999999999998

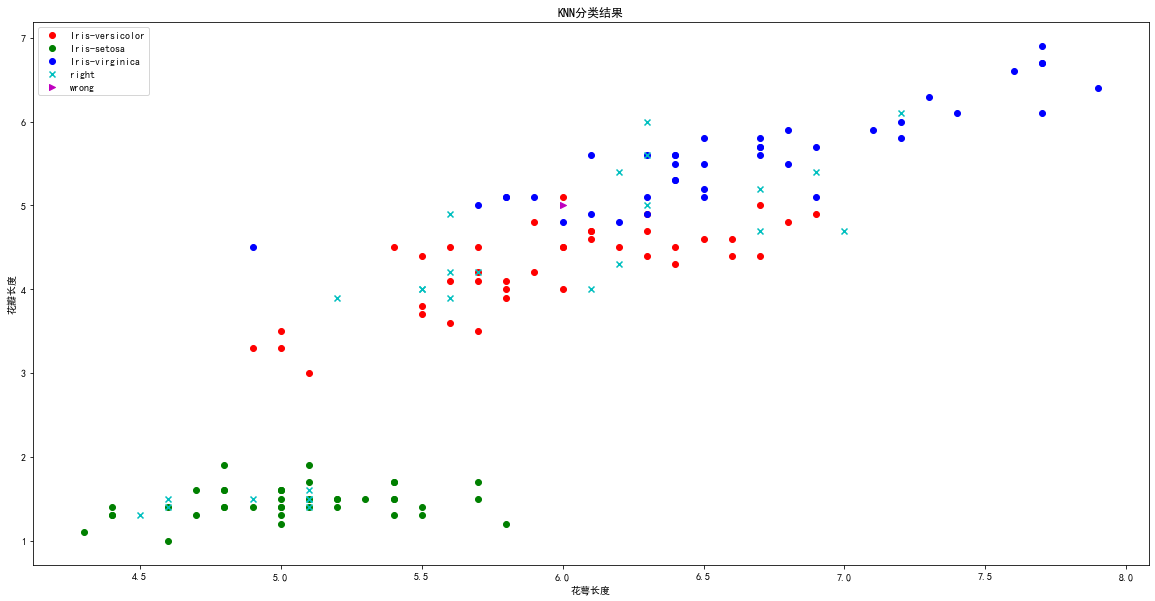
K: 10 accuracy: 0.9999999999999998

绘出折线图如下：



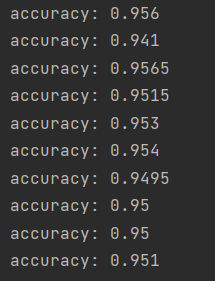
可以看到，K<=6时，分类准确率一直稳定在0.963，6~10之间准确率在不断波动，可见K值初始化选取的不同，对于分类点的分类结果影响较大。

分类结果可视化展示如下：

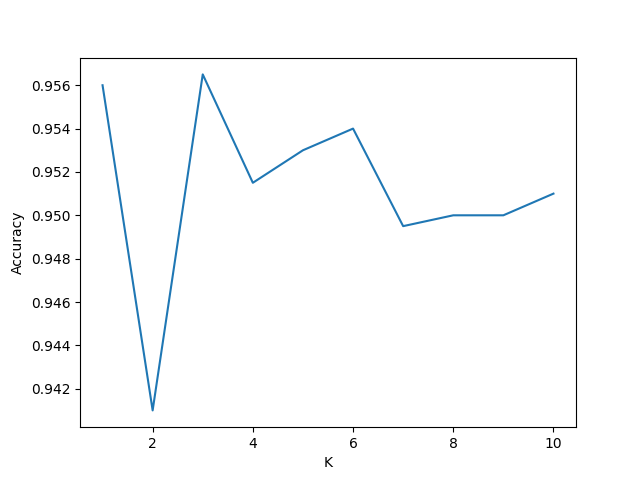


1. **MNIST数据集**

使用Mnist数据集，进行KNN算法运算，我们改变K的取值来研究不同K值下KNN在Iris数据集上的准确率，选取K值为1~10，得到分类准确率如下：



绘出折线图如下：



可以看到，采用Mnist数据集验证KNN的准确率，K的取值对分类准确率影响很大。1~10的K取值，分类准确率一直在波动变化，当K=3时，准确率最高为0.9565，K>3时小幅波动，趋于稳定。

因此，可得KNN中k值的选取策略：

* K值较小，则模型复杂度较高，容易发生过拟合，学习的估计误差会增大，预测结果对近邻的实例点非常敏感。
* K值较大可以减少学习的估计误差，但是学习的近似误差会增大，与输入实例较远的训练实例也会对预测起作用，使预测发生错误，k值增大模型的复杂度会下降。
* 在应用中，k值一般取一个比较小的值，通常采用交叉验证法来来选取最优的K值。

### 源代码附录

1.Iris数据集

1. **import** numpy as np
2. **import** pandas as pd
3. ```

6. ```python
7. data = pd.read\_csv(r"iris.arff.csv", header=0)
8. # data.head(10)
9. # data.tail(10)
10. **print**(data.sample(10))
11. data["class"] = data["class"].map({"Iris-versicolor":0,"Iris-setosa":1,"Iris-virginica":2}) # 类别名称映射为数字
12. # data = data.drop("Id",axis=1)  # 删除列
13. **print**(len(data))
14. **if** data.duplicated().any(): # 重复值
15. data.drop\_duplicates(inplace=True) #删除重复值
16. **print**(len(data))
17. data["class"].value\_counts()  # 查看各个类别的鸢尾花记录
18. ```
20. **class** KNN:
21. '''''使用KNN实现K近邻算法实现分类'''
22. **def** \_\_init\_\_(self, k):
23. '''''初始化
25. Parameters
26. -----
27. k:int
28. 邻居个位数
30. '''
31. self.k = k
32. **def** fit(self, X, y):
33. '''''训练
35. Parameeters
36. -----
37. X: 类数组类型，可以是List也可以是Ndarray，形状为： [样本数量,特征数量]
38. y: 类数组类型，形状为：[样本数量]
40. '''
41. self.X = np.asarray(X) #转换为ndarray类型
42. self.y = np.asarray(y)
43. **def** predict(self, X):
44. '''''对样本进行预测
45. Parameters:
46. X: 类数组类型，可以是List也可以是Ndarray，形状为： [样本数量,特征数量]
47. Returns:
48. 数组类型，预测结果
49. '''
50. X = np.asarray(X)
51. result = []
52. **for** x **in** X:
53. dis = np.sqrt(np.sum((x-self.X)\*\*2, axis=1)) # 对于测试机的每隔一个样本，一次与训练集的所有数据求欧氏距离
54. index = dis.argsort()# 返回排序结果的下标
55. index = index[:self.k] # 截取前K个
56. count = np.bincount(self.y[index]) # 返回数组中每个整数元素出现次数，元素必须是非负整数
57. result.append(count.argmax()) # 返回ndarray中值最大的元素所对应的索引，就是出现次数最多的索引，也就是我们判定的类别
58. **return** np.asarray(result)
59. **def** predict2(self, X):
60. '''''对样本进行预测，加入权重计算
61. Parameters:
62. X: 类数组类型，可以是List也可以是Ndarray，形状为： [样本数量,特征数量]
63. Returns:
64. 数组类型，预测结果
65. '''
66. X = np.asarray(X)
67. result = []
68. **for** x **in** X:
69. dis = np.sqrt(np.sum((x-self.X)\*\*2, axis=1)) # 对于测试机的每隔一个样本，一次与训练集的所有数据求欧氏距离
70. index = dis.argsort()# 返回排序结果的下标
71. index = index[:self.k] # 截取前K个
72. count = np.bincount(self.y[index], weights=1/dis[index]) # 返回数组中每个整数元素出现次数，元素必须是非负整数
73. result.append(count.argmax()) # 返回ndarray中值最大的元素所对应的索引，就是出现次数最多的索引，也就是我们判定的类别
74. **return** np.asarray(result)

77. ```python
78. # 提取每个类中鸢尾花数据
79. t0 = data[data["class"]==0]
80. t1 = data[data["class"]==1]
81. t2 = data[data["class"]==2]
82. # 打乱# 打乱每个类别数据
83. t0 = t0.sample(len(t0), random\_state=0)
84. t1 = t1.sample(len(t1), random\_state=0)
85. t2 = t2.sample(len(t2), random\_state=0)
86. # 分配训练集和数据集，axis=0表示按纵向方式拼接
87. train\_X = pd.concat([t0.iloc[:40, :-1], t1.iloc[:40, :-1], t2.iloc[:40, :-1]], axis=0)
88. train\_y = pd.concat([t0.iloc[:40, -1], t1.iloc[:40, -1], t2.iloc[:40, -1]], axis=0)
89. test\_X = pd.concat([t0.iloc[40:, :-1], t1.iloc[40:, :-1], t2.iloc[40:, :-1]], axis=0)
90. test\_y = pd.concat([t0.iloc[40:, -1], t1.iloc[40:, -1], t2.iloc[40:, -1]], axis=0)
91. ```

94. ```python
95. knn = KNN(k=3)
96. knn.fit(X=train\_X, y=train\_y)
97. result = knn.predict(test\_X)
98. # display(result)
99. # display(test\_y)
100. display(np.sum(result==test\_y))
101. display(np.sum(result==test\_y)/len(result))
102. ```

105. 26


109. 0.9629629629629629


113. ```python
114. #有权重的knn
115. result2 = knn.predict2(test\_X)
116. display(np.sum(result==test\_y))
117. display(np.sum(result==test\_y)/len(result))
118. ```

121. 26


125. 0.9629629629629629


129. ```python
130. # "Iris-versicolor":0,"Iris-setosa":1,"Iris-virginica":2
131. **import** matplotlib as mpl
132. **import** matplotlib.pyplot as plt
133. plt.figure(figsize=(20,10))
134. mpl.rcParams["font.family"] = 'SimHei' # 默认mpl不支持中文，设置一下支持
135. mpl.rcParams["axes.unicode\_minus"] = False # 设置中文字体是可以正常显示负号
136. plt.scatter(x=t0["sepallength"][:40], y=t0["petallength"][:40], color='r', label="Iris-versicolor")
137. plt.scatter(x=t1["sepallength"][:40], y=t1["petallength"][:40], color='g', label="Iris-setosa")
138. plt.scatter(x=t2["sepallength"][:40], y=t2["petallength"][:40], color='b', label="Iris-virginica")
139. right = test\_X[result == test\_y]
140. wrong = test\_X[result != test\_y]
141. plt.scatter(x=right["sepallength"], y=right["petallength"], color='c', label="right", marker="x")
142. plt.scatter(x=wrong["sepallength"], y=wrong["petallength"], color='m', label="wrong", marker=">")
143. plt.xlabel('花萼长度')
144. plt.ylabel('花瓣长度')
145. plt.title('KNN分类结果')
146. plt.legend(loc='best')
147. plt.show()
148. ```





155. ```python
156. # 调用类进行测试
157. # 用不同的k值进行测试
158. k\_range = range(1, 11)
159. accuracy = []  # 准确率
160. # start\_t = time.time() # 计算运行时间
161. **for** k **in** k\_range:
162. classifier = KNN(k=k)  # 对于不同的k值创建classifier
163. classifier.fit(X=train\_X, y=train\_y)
164. pred = classifier.predict(test\_X)
165. result = np.sum(( pred==test\_y)/len( pred))  # score-->accuracy
166. accuracy.append(result)
167. #     print(k)
168. **print**("K:",k,"     accuracy:", result)
169. # end\_t = time.time()
170. # print("total time:", end\_t - start\_t)
172. # 准确率 - K值图
173. **import** matplotlib.pyplot as plt
175. plt.plot(k\_range, accuracy)
176. plt.xlabel('K')
177. plt.ylabel('Accuracy')
178. plt.axis([0, 11, 0.9, 1])
179. plt.show()
180. ```
182. K: 1      accuracy: 0.9629629629629628
183. K: 2      accuracy: 0.9629629629629628
184. K: 3      accuracy: 0.9629629629629628
185. K: 4      accuracy: 0.9629629629629628
186. K: 5      accuracy: 0.9629629629629628
187. K: 6      accuracy: 0.9629629629629628
188. K: 7      accuracy: 0.9999999999999998
189. K: 8      accuracy: 0.9629629629629628
190. K: 9      accuracy: 0.9999999999999998
191. K: 10      accuracy: 0.9999999999999998
192. Mnist数据集

# 调用库  
import os  
import struct  
import numpy as np  
import heapq  
import time  
  
# 数据读取函数  
  
  
# 读取数据集的函数,返回图片的像素点灰度值数据和数字标签  
# 读取训练集数据  
def load\_mnist\_train(path, kind='train'):  
 labels\_path = os.path.join(path,  
 '%s-labels-idx1-ubyte'  
 % kind)  
 images\_path = os.path.join(path,  
 '%s-images-idx3-ubyte'  
 % kind)  
 with open(labels\_path, 'rb') as lbpath:  
 magic, n = struct.unpack('>II',  
 lbpath.read(8))  
 labels = np.fromfile(lbpath,  
 dtype=np.uint8)  
  
 with open(images\_path, 'rb') as imgpath:  
 magic, num, rows, cols = struct.unpack('>IIII',  
 imgpath.read(16))  
 images = np.fromfile(imgpath,  
 dtype=np.uint8).reshape(len(labels), 784)  
  
 return images, labels  
  
  
# 读取测试集数据  
def load\_mnist\_test(path, kind='t10k'):  
 labels\_path = os.path.join(path,  
 '%s-labels-idx1-ubyte'  
 % kind)  
 images\_path = os.path.join(path,  
 '%s-images-idx3-ubyte'  
 % kind)  
 with open(labels\_path, 'rb') as lbpath:  
 magic, n = struct.unpack('>II',  
 lbpath.read(8))  
 labels = np.fromfile(lbpath,  
 dtype=np.uint8)  
  
 with open(images\_path, 'rb') as imgpath:  
 magic, num, rows, cols = struct.unpack('>IIII',  
 imgpath.read(16))  
 images = np.fromfile(imgpath,  
 dtype=np.uint8).reshape(len(labels), 784)  
  
 return images, labels  
  
  
# 读取数据，打印数据属性  
# 读取数据  
path = r"MNIST"  
train\_set, train\_labels = load\_mnist\_train(path) # 训练集  
test\_set, test\_labels = load\_mnist\_test(path) # 测试集  
  
print(train\_set.shape)  
print(train\_labels.shape)  
print(test\_set.shape)  
print(test\_labels.shape)  
train\_size = train\_set.shape[0] # 训练样本数  
test\_size = test\_set.shape[0] # 测试样本数  
"""  
outputs:  
(60000, 784) # 784是28 \* 28,每一个图片中的像素点数，共有60000个训练集  
(60000,)  
(10000, 784) # 10000个测试集  
(10000,)  
"""  
# 数据归一化  
num\_pixel = 784 # 记录像素点数  
# 这里采用策略归一化到[0.0, 1.0]区间内  
train\_set = train\_set / 255.0  
test\_set = test\_set / 255.0  
# 采用部分数据进行测试，减少计算时间成本，检验合理性，验证算法是否正常工作  
# 先选择部分数据集进行测试  
num\_chosen\_train = 12000 # 选用12000个训练数据  
num\_chosen\_test = 2000 # 选用2000个测试数据  
  
# 随机选择下标  
train\_chosen\_index = np.random.choice(np.arange(train\_size), size=num\_chosen\_train, replace=False)  
test\_chosen\_index = np.random.choice(np.arange(test\_size), size=num\_chosen\_test, replace=False)  
  
# 随机选择出的训练图片  
train\_part\_images = train\_set[train\_chosen\_index]  
train\_part\_labels = train\_labels[train\_chosen\_index]  
  
# 随机选择出的测试图片  
test\_part\_images = test\_set[test\_chosen\_index]  
test\_part\_labels = test\_labels[test\_chosen\_index]  
# 创建KNNClassifier类  
"""  
为了充分利用numpy对矩阵计算的优化，将每一个测试集的1 \* 784的数据堆叠为len(x\_train) \* 784的矩阵，  
然后进行直接相减求二次幂，每一行的结果相加的方式更加快速地求得测试集与每一个训练集的欧几里得距离的平方大小；  
（不执行开根运算效果一样，减少运算量）  
"""  
  
  
# 分类器类实现  
class KNNClassifier():  
 def \_\_init\_\_(self, k=10):  
 self.k = k  
 self.x\_train = None  
 self.y\_train = None  
  
 # 传入训练数据  
 def fit(self, x\_train, y\_train):  
 self.x\_train = x\_train  
 self.y\_train = y\_train  
  
 def predict(self, x\_test):  
 labels\_pred = np.zeros(len(x\_test)) # 测试集待预测结果  
 for i in range(len(x\_test)):  
 # np.tile()对单个测试集数据进行堆叠，但是堆叠是在同一维度上  
 # 使用reshape将一维堆叠结果转化为与self.x\_train大小相同的矩阵  
 X = np.reshape(np.tile(x\_test[i], self.x\_train.shape[0]), (self.x\_train.shape[0], num\_pixel))  
 distance = np.sum(np.square(self.x\_train - X), axis=1) # 使用欧氏距离的平方作为“距离”度量  
 index = heapq.nsmallest(self.k, range(len(distance)), distance.take) # 找到distance中最小的k个值的下标  
 label = self.y\_train[index] # 待选的k个label  
 labels\_pred[i] = np.argmax(np.bincount(label)) # bincount()实现对标签出现次数的计算，比如5出现3次，那么返回np数组就有下标5对应数据3  
 return labels\_pred  
  
 def score(self, y\_pred, y\_test):  
 rate = np.sum(np.array(y\_pred == y\_test, dtype=np.int32))  
 return rate / len(y\_test) # / test\_length  
  
  
# 调用类进行测试  
# 用不同的k值进行测试  
k\_range = range(1, 11)  
accuracy = [] # 准确率  
# start\_t = time.time() # 计算运行时间  
for k in k\_range:  
 classifier = KNNClassifier(k=k) # 对于不同的k值创建classifier  
 classifier.fit(train\_part\_images, train\_part\_labels)  
 pred = classifier.predict(test\_part\_images)  
 result = classifier.score(pred, test\_part\_labels) # score-->accuracy  
 accuracy.append(result)  
 print("accuracy:", result)  
# end\_t = time.time()  
# print("total time:", end\_t - start\_t)  
  
"""  
outputs:  
accuracy: 0.9515  
accuracy: 0.9415  
accuracy: 0.9525  
accuracy: 0.952  
accuracy: 0.952  
accuracy: 0.9475  
accuracy: 0.9515  
accuracy: 0.9485  
accuracy: 0.9455  
accuracy: 0.946  
"""  
# 准确率 - K值图  
import matplotlib.pyplot as plt  
  
plt.plot(k\_range, accuracy)  
plt.xlabel('K')  
plt.ylabel('Accuracy')  
plt.show()