姓名：向建宇

学号：1801210687

1. 用letter-recognition数据集测试knn算法

根据knn算法的步骤，首先要将测试数据分成训练集和测试集，所以首先要解析文件，读入数据：

|  |
| --- |
| **def** file2matrix(filename): *#打开文件,此次应指定编码* fr = open(filename,**'r'**,encoding = **'utf-8'**) *#读取文件所有内容* arrayOLines = fr.readlines() *#针对有BOM的UTF-8文本，应该去掉BOM，否则后面会引发错误。* arrayOLines[0]=arrayOLines[0].lstrip(**'\ufeff'**) *#得到文件行数* numberOfLines = len(arrayOLines) *#返回的NumPy矩阵,解析完成的数据:numberOfLines行,16列* returnMat = np.zeros((numberOfLines,16)) *#返回的分类标签向量* classLabelVector = [] *#行的索引值* index = 0  **for** line **in** arrayOLines: *#s.strip(rm)，当rm空时,默认删除空白符(包括'\n','\r','\t',' ')* line = line.strip() *#使用s.split(str="",num=string,cout(str))将字符串根据','分隔符进行切片。* listFromLine = line.split(**','**) *#将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵* returnMat[index,:] = listFromLine[1:17] *#类别* classLabelVector.append(listFromLine[0])  index += 1  **return** returnMat, classLabelVector |

上面代码将文件读入并将特征存为一个16列的矩阵，26个英文字母类别也按照根矩阵的顺序对应存入。

第二步就是进行rnn算法了：

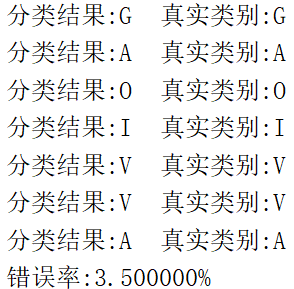
|  |
| --- |
| **def** classify0(inx, dataset, labels, k):  *# 计算距离* dist = np.sum((inx - dataset)\*\*2, axis=1)\*\*0.5  *# k个最近的标签* k\_labels = [labels[index] **for** index **in** dist.argsort()[0 : k]]  *# 出现次数最多的标签即为最终类别* label = collections.Counter(k\_labels).most\_common(1)[0][0]  **return** label |

如上所示，将测试集取出一行，然后和训练集每一行都计算出余旋距离，并选出k个最短距离，然后统计得到出现次数最多的类别

|  |
| --- |
| **def** datingClassTest():  *#打开的文件名* filename = **'letter-recognition.data'** *#将返回的特征矩阵和分类向量分别存储到datingDataMat和datingLabels中* datingDataMat, datingLabels = file2matrix(filename)  *#取所有数据的百分之十* hoRatio = 0.10  *#数据归一化,返回归一化后的矩阵,数据范围,数据最小值* normMat, ranges, minVals = autoNorm(datingDataMat)  *#获得normMat的行数* m = normMat.shape[0]  *#百分之十的测试数据的个数* numTestVecs = int(m \* hoRatio)  *#分类错误计数* errorCount = 0.0   **for** i **in** range(numTestVecs):  *#前numTestVecs个数据作为测试集,后m-numTestVecs个数据作为训练集* classifierResult = classify0(normMat[i,:], normMat[numTestVecs:m,:],  datingLabels[numTestVecs:m], 4)  print(**"分类结果:%s\t真实类别:%s"** % (classifierResult, datingLabels[i]))  **if** classifierResult != datingLabels[i]:  errorCount += 1.0  print(**"错误率:%f%%"** %(errorCount/float(numTestVecs)\*100)) |

测试集取了原本数据的前10%，后90%为训练集。

最终结果如图：



1. 用page-blocks.data数据集测试决策树算法

首先在测试之前，要先生成决策树，根据ppt和教材上的步骤：

|  |
| --- |
| 输入: 训练集D={(𝑥\_1, 𝑦\_1), (𝑥\_2, 𝑦\_2), ... , (𝑥\_𝑚, 𝑦\_𝑚),};  属性集A={𝑎\_1, 𝑎\_2, ... , 𝑎\_𝑑}  过程: 函数TreeGenerate(D, A)  1:生成结点node;  2: if D中样本全属于同一类别C then  3: 将node标记为C类叶结点; return  4: end if  5: if A = ∅ OR D中样本在A上取值相同 then  6: 将node标记为叶结点，其类别标记为D中样本数最多的类; return  7: end if  8: 从A中选择最优划分属性𝑎\_∗;  9: for 𝑎\_∗ 的每一个值 𝑎\_∗^𝑣 do  10: 为node生成一个分支; 令𝐷\_𝑣表示D中在𝑎\_∗上取值为𝑎\_∗^𝑣的样本子集;  11: if 𝐷\_𝑣 为空 then  12: 将分支结点标记为叶结点，其类别标记为D中样本最多的类; return  13: else  14: 以TreeGenerate(𝐷\_𝑣, A \ {𝑎\_∗})为分支结点  15: end if  16: end for  输出:以node为根结点的一棵决策树 |

老师给的例子只包括了对离散值的处理，所以在对连续值时𝑎\_∗要取一个中间值，并以此为结点划分成大于它和小于它两个分支，

|  |
| --- |
| **def** generate\_decision\_tree(data\_set, attribute\_label):  label\_list = [entry[-1] **for** entry **in** data\_set]   **if** label\_list.count(label\_list[0]) == len(label\_list): *# 如果所有的数据都属于同一个类别，则返回该类别* **return** label\_list[0]   **if** len(data\_set[0]) == 1: *# 如果数据没有属性值数据，则返回该其中出现最多的类别作为分类* **return** most\_voted\_attribute(label\_list)   best\_attribute\_index, best\_split\_point = attribute\_selection\_method(data\_set)   best\_attribute = attribute\_label[best\_attribute\_index]   decision\_tree = {best\_attribute: {}}   **del** (attribute\_label[best\_attribute\_index]) *# 找到最佳划分属性后需要将其从属性名列表中删除* **"""   如果best\_split\_point为空，说明此时最佳划分属性的类型为离散值，否则为连续值   """   if** best\_split\_point == **None**:   attribute\_list = [entry[best\_attribute\_index] **for** entry **in** data\_set]   attribute\_set = set(attribute\_list)   **for** attribute **in** attribute\_set: *# 属性的各个值* sub\_labels = attribute\_label[:]   decision\_tree[best\_attribute][attribute] = generate\_decision\_tree(   split\_data\_set(data\_set, best\_attribute\_index, attribute, continuous=**False**), sub\_labels)   **else**:   **"""   最佳划分属性类型为连续值，此时计算出的最佳划分点将数据集一分为二，划分字段取名为<=和>   """** sub\_labels = attribute\_label[:]   decision\_tree[best\_attribute][**"<="** + str(best\_split\_point)] = generate\_decision\_tree(   split\_data\_set(data\_set, best\_attribute\_index, best\_split\_point, **True**, 0), sub\_labels)   sub\_labels = attribute\_label[:]   decision\_tree[best\_attribute][**">"** + str(best\_split\_point)] = generate\_decision\_tree(   split\_data\_set(data\_set, best\_attribute\_index, best\_split\_point, **True**, 1), sub\_labels)   **return** decision\_tree |

其中最重要的一步是找到划分点，即在划分结点的时候要通过香农熵和信息增益来得到最佳划分，下面是找到最佳划分属性的过程

|  |
| --- |
| """  通过信息增益比来计算最佳划分属性  属性分为离散值和连续值两种情况，分别对两种情况进行相应计算  """  def attribute\_selection\_method(data\_set):  num\_attributes = len(data\_set[0]) - 1 # 属性的个数，减1是因为去掉了标签  info\_D = calc\_info\_D(data\_set) # 香农熵  max\_grian\_rate = 0.0 # 最大信息增益比  best\_attribute\_index = -1  best\_split\_point = None  continuous = False  for i in range(num\_attributes):  attribute\_list = [entry[i] for entry in data\_set] # 求属性列表，此时为连续值  info\_A\_D = 0.0 # 特征A对数据集D的信息增益  split\_info\_D = 0.0 # 数据集D关于特征A的值的熵  #if attribute\_list[0] not in set(['M', 'F', 'I']): #判断离散值  #continuous = True  """  属性为连续值，先对该属性下的所有离散值进行排序  然后每相邻的两个值之间的中点作为划分点计算信息增益比，对应最大增益比的划分点为最佳划分点  由于可能多个连续值可能相同，所以通过set只保留其中一个值  """  if continuous == True:  attribute\_list = sort(attribute\_list)  temp\_set = set(attribute\_list) # 通过set来剔除相同的值  attribute\_list = [attr for attr in temp\_set]  split\_points = []  for index in range(len(attribute\_list) - 1):  # 求出各个划分点  split\_points.append((float(attribute\_list[index]) + float(attribute\_list[index + 1])) / 2)  for split\_point in split\_points: # 对划分点进行遍历  info\_A\_D = 0.0  split\_info\_D = 0.0  for part in range(2): # 最佳划分点将数据一分为二，因此循环2次即可得到两段数据  sub\_data\_set = split\_data\_set(data\_set, i, split\_point, True, part)  prob = len(sub\_data\_set) / float(len(data\_set))  info\_A\_D += prob \* calc\_info\_D(sub\_data\_set)  split\_info\_D -= prob \* log(prob, 2)  if split\_info\_D == 0:  split\_info\_D += 1  """  由于关于属性A的熵split\_info\_D可能为0，因此需要特殊处理  常用的做法是把求所有属性熵的平均，为了方便，此处直接加1  """  grian\_rate = (info\_D - info\_A\_D) / split\_info\_D # 计算信息增益比  if grian\_rate > max\_grian\_rate:  max\_grian\_rate = grian\_rate  best\_split\_point = split\_point  best\_attribute\_index = i  print([best\_attribute\_index, best\_split\_point])  else: # 划分属性为离散值  attribute\_list = [entry[i] for entry in data\_set] # 求属性列表  attribute\_set = set(attribute\_list)  for attribute in attribute\_set: # 对每个属性进行遍历  sub\_data\_set = split\_data\_set(data\_set, i, attribute, False)  prob = len(sub\_data\_set) / float(len(data\_set))  info\_A\_D += prob \* calc\_info\_D(sub\_data\_set)  split\_info\_D -= prob \* log(prob, 2)  if split\_info\_D == 0:  split\_info\_D += 1  grian\_rate = (info\_D - info\_A\_D) / split\_info\_D # 计算信息增益比  if grian\_rate > max\_grian\_rate:  max\_grian\_rate = grian\_rate  # print(max\_grian\_rate)  best\_attribute\_index = i  best\_split\_point = None # 如果最佳属性是离散值，此处将分割点置为空留作判定  return best\_attribute\_index, best\_split\_point |

其中会调用计算数据集D的香农熵

|  |
| --- |
| **def** calc\_info\_D(data\_set):  num\_entries = len(data\_set)   label\_nums = {} *# 为每个类别建立字典，value为对应该类别的数目* **for** entry **in** data\_set:   label = entry[-1]   **if** label **in** label\_nums.keys():   label\_nums[label] += 1   **else**:   label\_nums[label] = 1   info\_D = 0.0   **for** label **in** label\_nums.keys():  prob = float(label\_nums[label]) / num\_entries   info\_D -= prob \* log(prob, 2)   **return** info\_D |

以及在生成决策树时遇到递归返回情况下多数表决：返回标签列表中数量最大的类

|  |
| --- |
| **def** most\_voted\_attribute(label\_list):  label\_nums = {}   **for** label **in** label\_list:   **if** label **in** label\_nums.keys():   label\_nums[label] += 1   **else**:   label\_nums[label] = 1   sorted\_label\_nums = sorted(label\_nums.items(), key=operator.itemgetter(1), reverse=**True**)   **return** sorted\_label\_nums[0][0] |

最后通过递归生成一颗用字典对应的树后，进行测试：

|  |
| --- |
| **def** decision\_tree\_predict(decision\_tree, attribute\_labels, one\_test\_data):  first\_key = list(decision\_tree.keys())[0]   second\_dic = decision\_tree[first\_key]   attribute\_index = attribute\_labels.index(first\_key)   res\_label = **None   for** key **in** second\_dic.keys(): *# 属性分连续值和离散值，连续值对应<=和>两种情况* **if** key[0] == **'<'**:   value = float(key[2:])   **if** float(one\_test\_data[attribute\_index]) <= value:   **if** type(second\_dic[key]).\_\_name\_\_ == **'dict'**:   res\_label = decision\_tree\_predict(second\_dic[key], attribute\_labels, one\_test\_data)   **else**:   res\_label = second\_dic[key]   **elif** key[0] == **'>'**:   *# print(key[1:])* value = float(key[1:])   **if** float(one\_test\_data[attribute\_index]) > value:   **if** type(second\_dic[key]).\_\_name\_\_ == **'dict'**:   res\_label = decision\_tree\_predict(second\_dic[key], attribute\_labels, one\_test\_data)   **else**:   res\_label = second\_dic[key]     **else**:   **if** one\_test\_data[attribute\_index] == key:   **if** type(second\_dic[key]).\_\_name\_\_ == **'dict'**:   res\_label = decision\_tree\_predict(second\_dic[key], attribute\_labels, one\_test\_data)   **else**:   res\_label = second\_dic[key]   **return** res\_label |

其中也划分了连续值和离散值，判断依据为key有没有包含>和<号，以及等于标签三种情况，然后再往下进行判断。

测试结果如图：

C:\Users\xiang\AppData\Local\Temp\1538657704(1).png

其中训练集和测试集的划分使用了sklearn.cross\_validation 中的train\_test\_split方法，每次的测试集选择完全随机，经过多次测试，大概正确率稳定在70%左右

F:\Anaconda3

F:\Anaconda3\Library\mingw-w64\bin

F:\Anaconda3\Library\bin

F:\Anaconda3\Scripts