CSCI964 Computational Intelligence: Lab #6

Mei Wangzhihui 2019124044

Task 1

```
def euler_distance(point1: np.ndarray, point2: list) -> float:
   """
   计算两点之间的欧拉距离，支持多维
   """
   distance = 0.0
   for a, b in zip(point1, point2):
      distance += math.pow(a - b, 2)
   return math.sqrt(distance)


class ClusterNode(object):
   def __init__(self, vec, left=None, right=None, distance=-1, id=None, count=1):
      """
      :param vec: 保存两个数据聚类后形成新的中心
      :param left: 左节点
      :param right: 右节点
      :param distance: 两个节点的距离
      :param id: 用来标记哪些节点是计算过的
      :param count: 这个节点的叶子节点个数
      """
      self.vec = vec
      self.left = left
      self.right = right
      self.distance = distance
      self.id = id
      self.count = count


class Hierarchical(object):
   def __init__(self, k=1):
      assert k > 0
      self.k = k
      self.labels = None

   def fit(self, x):
      nodes = [ClusterNode(vec=v, id=i) for i, v in enumerate(x)]
      distances = {}
      point_num, future_num = np.shape(x) # 特征的维度
      self.labels = [-1] * point_num
      currentclustid = -1
      while len(nodes) > self.k:
         min_dist = math.inf
         nodes_len = len(nodes)
         closest_part = None # 表示最相似的两个聚类
         for i in range(nodes_len - 1):
            for j in range(i + 1, nodes_len):
               # 为了不重复计算距离，保存在字典内
               d_key = (nodes[i].id, nodes[j].id)
               if d_key not in distances:
```

```
                        distances[d_key] = euler_distance(nodes[i].vec, nodes[j].vec)
                    d = distances[d_key]
                    if d < min_dist:
                        min_dist = d
                        closest_part = (i, j)
                        # 合并两个聚类
            part1, part2 = closest_part
            node1, node2 = nodes[part1], nodes[part2]
            new_vec = [(node1.vec[i] * node1.count + node2.vec[i] * node2.count) /
                (node1.count + node2.count) for i in range(future_num)]
            new_node = ClusterNode(vec=new_vec, left=node1, right=node2,
                distance=min_dist, id=currentclustid, count=node1.count + node2.count)
            currentclustid -= 1
            del nodes[part2], nodes[part1] # 一定要先del索引较大的
            nodes.append(new_node)
        self.nodes = nodes
        self.calc_label()

    def calc_label(self):
        """
        调取聚类的结果
        """
        for i, node in enumerate(self.nodes):
            # 将节点的所有叶子节点都分类
            self.leaf_traversal(node, i)

    def leaf_traversal(self, node: ClusterNode, label):
        """
        递归遍历叶子节点
        """
        if node.left == None and node.right == None:
            self.labels[node.id] = label
        if node.left:
            self.leaf_traversal(node.left, label)
        if node.right:
            self.leaf_traversal(node.right, label)


def setData(filename):
    #生成num个随机数据
    Data = np.loadtxt(filename, delimiter=',', usecols=(0, 1, 2, 3))
    return Data
```

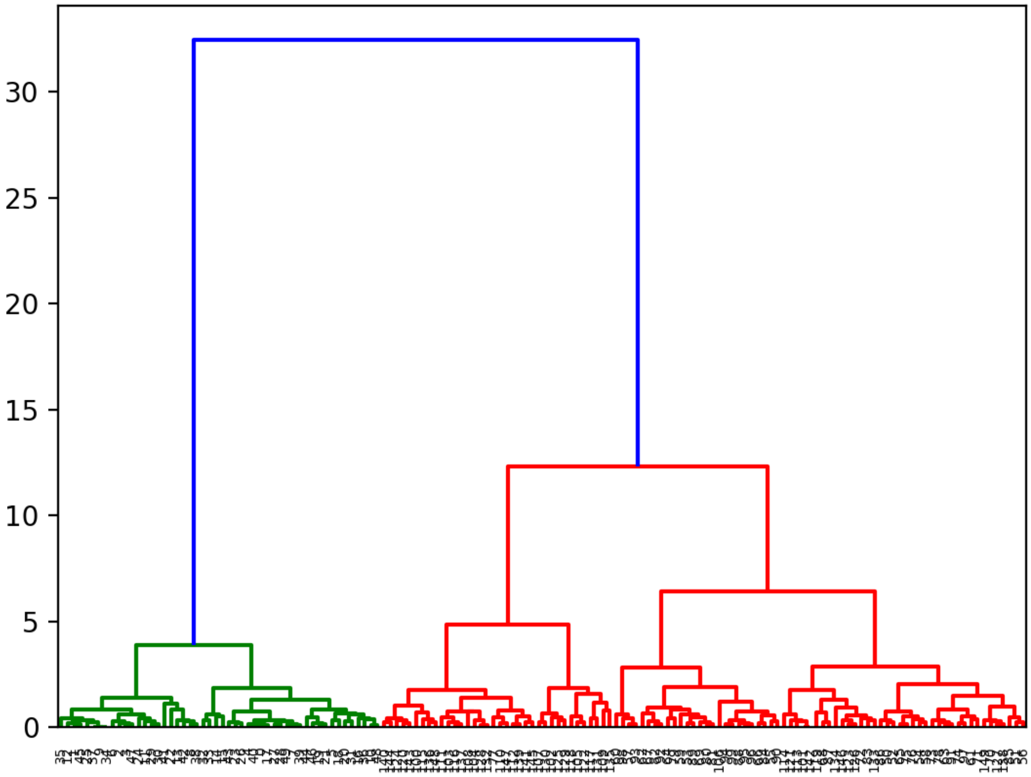

Figure 1: The Clustering plot

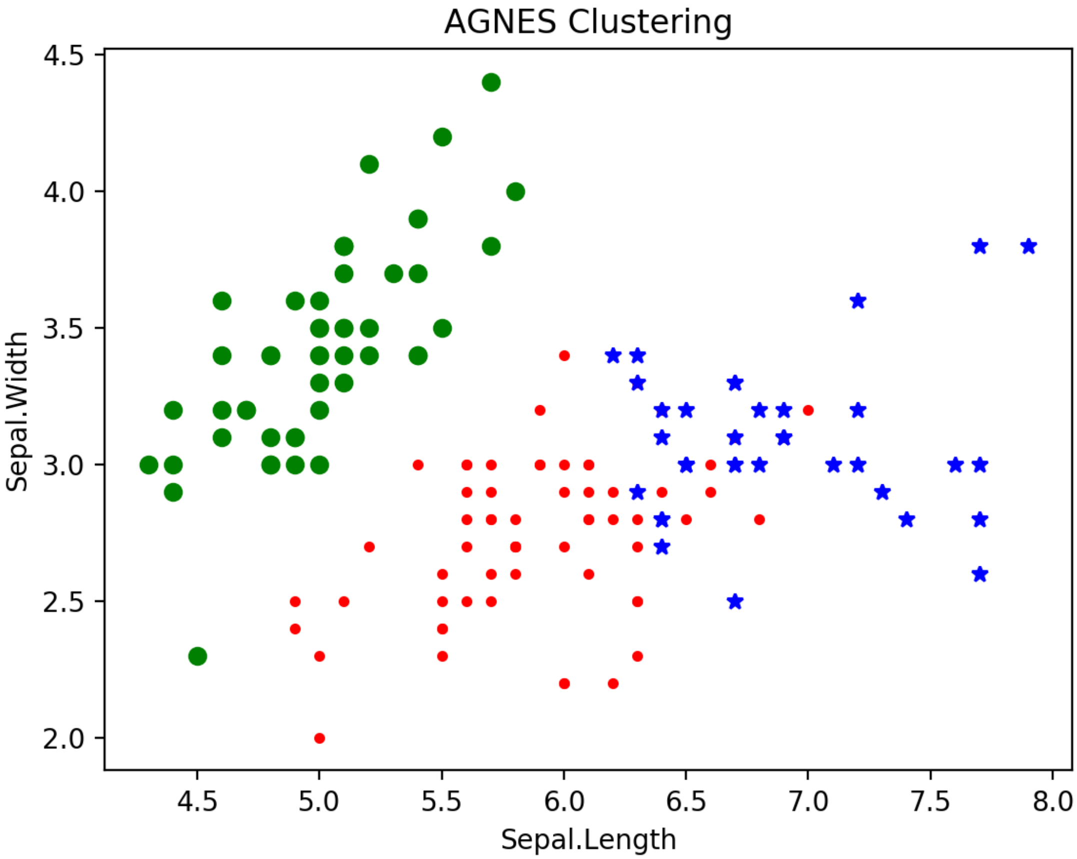

Figure 2: The Clustering graph