人工智能实验lab2 实验报告

学号：17341190 专业：计算机科学 姓名：叶盛源

决策树

一、算法原理

决策树本质是上是一系列规则队数据进行分类的过程。树中每个节点代表了一种属性，每个分支代表这个属性的不同值，叶子结点代表了最终分类的结果。在训练决策树时，最开始所有数据集中在根节点，随着选取节点和分支，将数据不断的划分成满足条件的子集，直到达到一个临界条件时停止划分。

决策树可以使用递归的方法构建，遇到临界条件则回溯，临界条件可以分为以下几种不同的情况：

1. 当前节点的数据样本全部属于同一个标签
2. 属性集为空集，标记为当前接单出现最多的类
3. 数据集为空集，标记为父节点中出现最多的类

我们肯定希望可以选择一个区分标签程度最大的节点作为根的节点，这样才能更好的划分数据集，使得决策树的分类效果更显著，因此如何选取结点的属性值很重要。

ID3决策树

ID3是一种基于信息增益选择节点的决策树算法。在选取节点的时候，ID3要遍历所有可以选取的属性，分别计算每个属性的信息熵，还要计算没有选取属性前的信息熵，将选取前的信息熵和选取后的信息熵做差，得到的就是信息增益。差的大小就代表了引入这个属性做了分类后，要判断标签的不确定性下降的幅度，如果信息增益大，说明引入这个属性后不确定性下降更多，因此从所有属性中选取信息增益最大的一项作为当前节点的属性，然后删去这个属性（避免重复）并向不同的分支递归构造决策树。

计算引入节点前的经验熵：

$$H(D)=-\Sigma_{d\in D}p(d)logp(d)$$

接着对每个特征A计算对数据集D的条件熵：

$$H(D)=-\Sigma_{a\in A}p(a)H(D|A=a)$$

然后我们可以利用上面的值做差计算信息增益大小：

$$g(D,A)=H(D)-H(D|A)$$

最后根据信息增益大小来选取决策树节点

C4.5决策树

C4.5决策树是ID3算法的一个改进，因为考虑到ID3这种选择节点的方法对于分支数更多的节点更有优势，于是C4.5就通过归一化分支的数量来解决这个问题，即在求属性A的信息增益后，再求关于属性A的信息熵，最后做除法的到信息增益率，用信息增益率进行比较。

计算属性A的信息熵

$$splitInfo(D,A)=-\sum_{j=1}\frac{|D_j|}{|D|}log(\frac{|D_j|}{|D|})$$

求信息增益率

$$gRatio(D,A)=(H(D)-H(D|A))/splitInfo(D,A)$$

最后选取信息增益率最大的作为决策树的节点

CART决策树

CART的逻辑和前两者相同，不过它并不是使用信息熵，而是使用了一个经济学的指标Gini系数来作为决策依据，选择Gini指数最小的特征来分离数据集递归构建决策树。

计算特征A条件下数据集D的Gini系数

$$gini(D,A)=\sum_{j=1}^{v}p(A_j)\times gini(D_j|A=A_j)$$

其中：

$$gini(D_j|A=A_j)=\sum_{i=1}^{n}p_i(1-p_i)=1-\sum_{i=1}^{n}p_i^2$$

因为gini系数计算的时候没有像信息增益的时候进行取反，因此是选取最小的Gini系数的作为节点，但其实本质上和信息增益的方法类似。

剪枝

决策树在训练完后可以进行剪枝操作来提升泛化能力，防止过拟合。剪枝包括两种：

1. 预剪枝：在决策树构建的时候进行，对当前节点，先做判断是否还需要继续划分，如果划分后决策树在验证集上的准确率没有提高，则无需划分。
2. 后剪枝：先生成完整的决策树，再从底向上对非叶节点考察，如果将它的叶子剪掉，将它变成叶节点，决策树在验证集上的准确率不降低，则将它变成叶节点

二、 伪代码

```
Procedure Decision_tree(data,k)
// input: data是文本的字符串组成列表
//        k是kfold要分的份数i
```

```
// return: 返回决策树
  // 分解数据集为训练集和验证集
  train_set,validation_set = k_fold(dataSet,k)

  //统计每个属性存在的子属性来构建节点的分支
  feature_dic = count_feature_dic(dataSet,labelSet)

  // 递归构建决策树
  root = createTree(train_set,feature_dic)

  // 在验证集上计算准确度
  res = validation(validation_set,root)

  // 输出决策树准确率结果
  print("对数据集进行%s折后的正确率为%s"%(k,res))


Procedure createTree(train_set,feature_dic)
// input: train_set节点的数据量
//        feature_dic可以使用的属性
//        root 当前根节点
// return: 返回决策树
  //选取当前最好的属性
  bestFeature = chooseBestFeature(train_set)
  available_feature.remove(bestFeature)// 从现有属性移除选择的属性

  // 从子特征字典中创建分支，递归建树
  for subfeature in feature_dic[bestFeature]:
    split_dataSet=splitDataSet(dataSet,bestFeature,subfeature)
    branch[subfeature]=createTree(train_set,feature_dic)

  // 返回当前创建的节点
  return DecisionNode(feature=bestFeature,branch=branch)
```

三、 代码展示

本次编程采用模块化编程，分为以下多个部分：

读取文件函数和对数据集和根据特定条件划分的函数：

```
def readDataSet():
    """
    导入数据
    """
    # 对数据进行处理
    dataSet = pd.read_csv('../data/lab2_dataset/car_data.csv')
```

```
    # print(dataSet.values)
    labelSet = list(dataSet.columns.values)# 读取属性名
    dataSet = list(dataSet.values) # 读取数据变成列表
    return dataSet, labelSet


def splitDataSet(dataSet, index, subfeature):
    '''
    根据选定的属性划分数据集
    :return:
    '''
    DataSet = []
    for record in dataSet:
        # 将相同数据特征的提取出来
        if record[index] == subfeature:
            DataSet.append(record)
    return DataSet
```

使用类来定义决策树的节点。

```
class DecisionNode(object):
    """
    决策树的节点
    """
    def __init__(self, label=None,feature=None,branch=None):
        self.feature = feature         # 当前节点对应的属性标签 众数
        self.label = label       # 保存的是针对当前分支的结果，有值则表示该点是叶子节点
        self.branch = branch             # 分支的字典
```

提前统计好每个属性有几个子属性。如果不提前统计，到后面划分了数据再统计可能会导致有一些特征缺失。

```
def count_feature_dic(dataSet,labelSet):
    '''
    计算特征的字典,格式为特征在labelSet的下标作为key sub特征作为values
    '''
    feature_num = len(labelSet)-1  # 属性的个数 不算标签
    feature_dic={}
    for i in range(0,feature_num):
        featureSet = set()
        for record in dataSet:
            featureSet.add(record[i]) # 获取当前特征所有可能的取值
        feature_dic[i] = featureSet

    return feature_dic
```

计算信息熵。有两种不同的信息熵需要计算，一种是label的信息熵，一种是属性的信息熵，属性的信息熵在C4.5的决策树中用来消除分支数量的影响。

```
def cal_label_InforEntropy(subdataset):
    '''
    计算当前数据集的信息熵值
    :param subdataset: 最后一列为标签值，其他为属性值
    :return: 返回信息熵的结果
    '''
    total = len(subdataset)
    labelCounts = {} # 统计各个label的数量
    for record in subdataset:
        currentLabel = record[-1]
        labelCounts[currentLabel]=labelCounts.get(currentLabel,0)+1

    InforEntropy = 0.0
    for item in labelCounts.items():
        prob = float(item[1]) / total
        if prob!=0:
            # 如果是0的时候就log0=0
            InforEntropy -= prob * np.log2(prob)
    return InforEntropy


def cal_feature_InforEntropy(subdataset,index):
    '''
        计算当前数据集某一个属性的信息熵值
        用于C4.5的决策树中
        :param subdataset: 最后一列为标签值，其他为属性值
        :return: 返回信息熵的结果
        '''
    total = len(subdataset)
    featureCounts = {}  # 统计各个subfeature的数量
    for record in subdataset:
        currentLabel = record[index]
        featureCounts[currentLabel] = featureCounts.get(currentLabel, 0) + 1

    InforEntropy = 0.0
    for item in featureCounts.items():
        prob = item[1] / total
        if prob != 0:
            # 如果是0的时候就0*log0=0
            InforEntropy -= prob * np.log2(prob)
    return InforEntropy
```

挑选最好的特征，可以通过传递参数来选择使用哪一种决策的方法，包括ID3，C4.5和Gini指数三种方法。

```
def chooseBestFeature(dataSet,feature_dic,available_feature,method="ID3"):
    '''
    根据前面的辅助函数选出最优的特征并
    作为当前子树的根结点
    '''
    if method=="ID3":
        total=len(dataSet)# 总的数据条数
        rootEntropy = cal_label_InforEntropy(dataSet) # 引入信息前的信息熵
        InforGain_list=[]
        for feature in available_feature:
            curInforEntropy = 0.0 # 当前特征计算出来的信息熵的值
            for subfeature in feature_dic[feature]:
                subDataSet = splitDataSet(dataSet, feature, subfeature)
                # 特征为i的数据集占总数的比例
                prob = len(subDataSet) / total
                curInforEntropy += prob * cal_label_InforEntropy(subDataSet)
            InforGain_list.append((rootEntropy - curInforEntropy,feature))

        heapq.heapify(InforGain_list)
        # print(InforGain_list)
        return heapq.nlargest(1,InforGain_list)[0][1]

    elif method == "C4.5":
        total = len(dataSet)  # 总的数据条数
        rootEntropy = cal_label_InforEntropy(dataSet)  # 引入信息前的信息熵
        InforGain_list = []
        for feature in available_feature:
            curInforEntropy = 0.0  # 当前特征计算出来的信息熵的值
            for subfeature in feature_dic[feature]:
                subDataSet = splitDataSet(dataSet, feature, subfeature)
                # 特征为i的数据集占总数的比例
                prob = len(subDataSet) / total
                curInforEntropy += prob * cal_label_InforEntropy(subDataSet)

            # 计算feature的信息熵:
            splitInfo = cal_feature_InforEntropy(dataSet,feature)
            if splitInfo==0:
                continue
            InforGain_list.append(((rootEntropy - curInforEntropy)/splitInfo,
feature))

        heapq.heapify(InforGain_list)
        # print(InforGain_list)
        return heapq.nlargest(1, InforGain_list)[0][1]

    elif method =="Gini":
```

```
        total = len(dataSet)
        InforGini_list=[]
        for feature in available_feature:
            curInfoGini=0
            subfeature_count_dic = countsubfeature(dataSet, feature) # 统计每个
子特征出现次数 和对应标签的出现次数
            for item in subfeature_count_dic.items():
                feature_prob=item[1][0]/total # 属性的概率值
                label_prob=0 # 统计标签的概率平方和
                for label in item[1][1].items():
                    label_prob+=np.power(label[1]/item[1][0],2)
                curInfoGini+=feature_prob*(1-label_prob)

        InforGini_list.append((curInfoGini , feature))

        heapq.heapify(InforGini_list)
        # print(InforGain_list)
        return heapq.nsmallest(1, InforGini_list)[0][1] # 选取最小的作为Gini指标选
出的结果
```

使用递归的方法来构建决策树模型，每次先选择最优属性，然后划分数据集进行递归构建决策树的模型。

```
def createTree(dataSet,parent_label,available_feature,feature_dic):
    """
    构造决策树
    """
    # 收集最后一列的标签值
    classList = [record[-1] for record in dataSet]
    # 如果数据集为空集，则节点的属性取父节点的值
    if len(dataSet) == 0:
        return DecisionNode(label=parent_label,feature=None, branch=None)
    # 当类别与属性完全相同时停止
    if classList.count(classList[0]) == len(classList):
        return DecisionNode(label=classList[0],feature=None, branch=None)
    # 当没有特征值时，直接返回数量最多的属性作为节点的值
    if (len(available_feature) == 0):
        return DecisionNode(label=max(classList,key=classList.count),
feature=None, branch=None)

    # 选出最好的特征 对应数据集的列index
    bestFeature =
chooseBestFeature(dataSet,feature_dic,available_feature,method="ID3")
    available_feature.remove(bestFeature) # 删去用掉的最好特征
    branch={}
    parent_label=max(classList,key=classList.count) # 求出当前节点类别数量最多的一
者传到子节点
```

```
    for subfeature in feature_dic[bestFeature]:
        split_dataSet=splitDataSet(dataSet,bestFeature,subfeature) # 根据子特征
的值提取数据

 branch[subfeature]=createTree(split_dataSet,parent_label,available_feature[:]
,feature_dic)

    return DecisionNode(feature=bestFeature,label=parent_label,branch=branch)
```

在验证集上检验实验结果，使用kfold的方法选取训练集和验证集。kfold是将数据集分成k份，选取第i份作为验证集其他作为训练集，最后就i取所有值下的均值作为最终的正确率。

```
def k_fold(dataSet,k,i):
    '''
    对数据集进行训练集和验证集的划分
    :param dataSet: 数据集
    :param k: 划分成k个部分
    :param i: 选取第i个部分为验证集
    :return: 返回训练和验证集的数据
    '''
    total = len(dataSet)
    step_len=total//k # 求出每一份的长度
    val_begin=i*step_len
    val_end=val_begin+step_len
    return dataSet[:val_begin]+dataSet[val_end:],dataSet[val_begin:val_end]



def validation(validation_dataset,root):
    count = 0
    for i,record in enumerate(validation_dataset):
        cur=root
        while cur.branch != None:
            cur = cur.branch[record[cur.feature]]
        if cur.label==record[-1]:
            count+=1
    return count/len(validation_dataset)
```

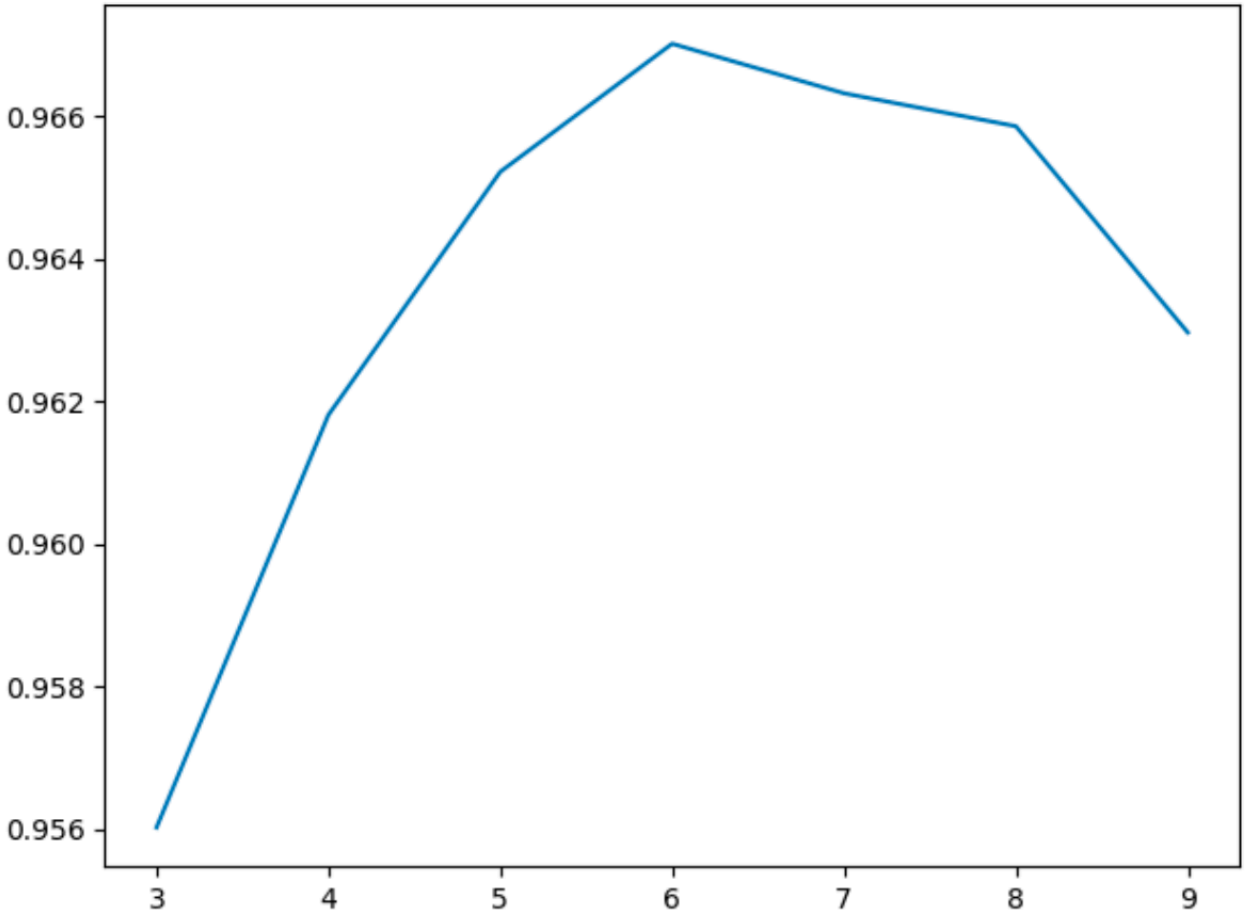
四、 实验结果以及分析：

ID3决策树

调整k_fold验证的k值，划分不同数量的验证集和训练集进行实验：

```
对数据集进行3折后的正确率为0.9560185185185185
对数据集进行4折后的正确率为0.9618055555555556
对数据集进行5折后的正确率为0.9652173913043478
对数据集进行6折后的正确率为0.967013888888889
对数据集进行7折后的正确率为0.9663182346109176
对数据集进行8折后的正确率为0.9658564814814815
对数据集进行9折后的正确率为0.9629629629629629
```

收集每个k和对应的正确率作出折线图可得：



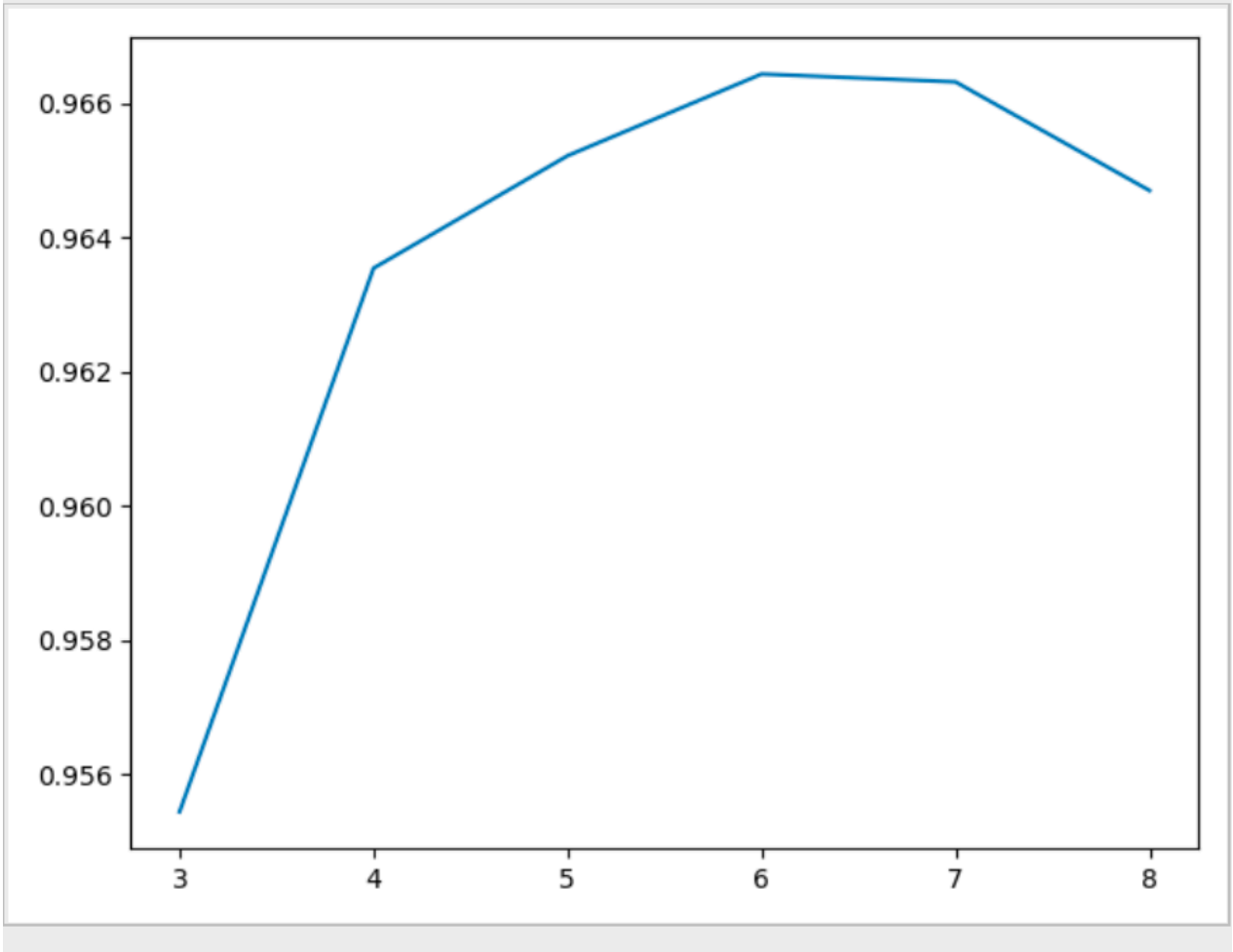
由于数据量不大，如果k过大，验证集的数量就会太少，得到的验证集准确率的随机性比较大。 但如果k过小，训练集数量少，也可能导致验证集上的表现不佳，因此这里选择了3到9作图，可以看到，准确率呈现先上升后下降的趋势。而且决策树准确率普遍在0.96以上，可见这个决策树的效果还是不错的。

C4.5决策树

调整k_fold验证的k值，划分不同数量的验证集和训练集进行实验：

```
对数据集进行3折后的正确率为0.9554398148148149
对数据集进行4折后的正确率为0.9635416666666667
对数据集进行5折后的正确率为0.9652173913043478
对数据集进行6折后的正确率为0.9664351851851852
对数据集进行7折后的正确率为0.9663182346109176
对数据集进行8折后的正确率为0.9646990740740741
对数据集进行9折后的正确率为0.9618055555555556
```

收集每个k和对应的正确率作出折线图可得：



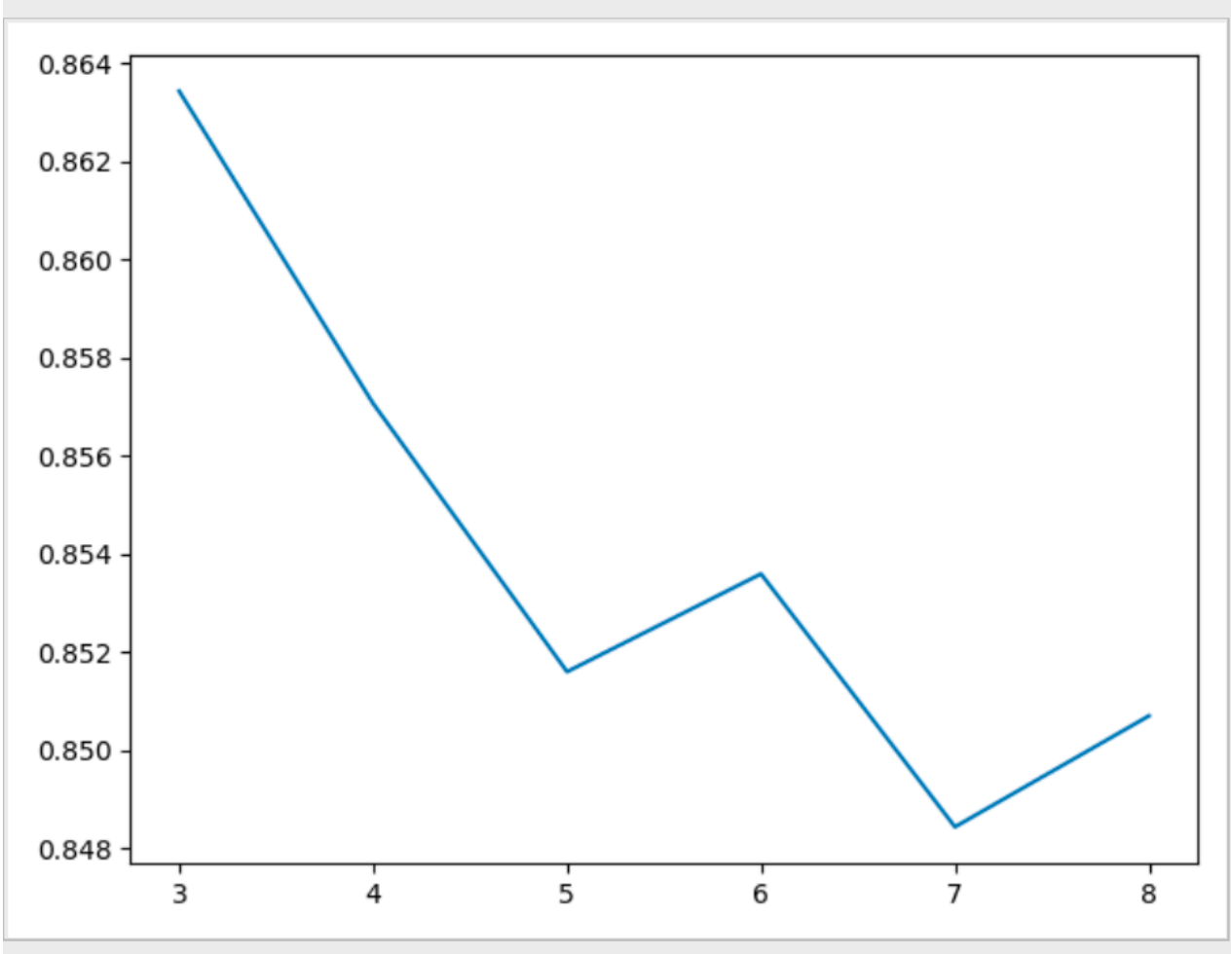
ID3决策树的折线图和C4.5的折线图趋势相同，都是先上升后下降，而且准确度十分的相近，这是因为在本次试验中每个属性的分支数不是很多，而且数量平均，因此归一化的方法的效果并不是十分的明显，甚至在一些k值的时候准确度反而比ID3的方法更低。

CART决策树

调整k_fold验证的k值，划分不同数量的验证集和训练集进行实验：

```
对数据集进行3折后的正确率为0.8634259259259259
对数据集进行4折后的正确率为0.8570601851851852
对数据集进行5折后的正确率为0.8515942028985508
对数据集进行6折后的正确率为0.8535879629629629
对数据集进行7折后的正确率为0.8484320557491288
对数据集进行8折后的正确率为0.8506944444444444
对数据集进行9折后的正确率为0.8489583333333333
```

收集每个k和对应的正确率作出折线图可得：



可以看到Gini指数的方法准确率普遍较低，在85左右，并且随着k值的增大，总体趋势是准确度不断下降。输出决策树后可以看到是因为在一些节点上的选择和前两种结果不同，所以在本次实验的数据集上，CART的表现是相对较差的，而Gini系数选取节点的方法在本次实验中也不如信息增益的方法。

五、 思考题

- 决策树有哪些避免过拟合的方法：

  决策树在训练完后可以进行剪枝操作来提升泛化能力，防止过拟合。剪枝包括两种：

  1）预剪枝：在决策树构建的时候进行，对当前节点，先做判断是否还需要继续划分，如果划分后决策树在验证集上的准确率没有提高，则无需划分。

  2）后剪枝：先生成完整的决策树，再从底向上对非叶节点考察，如果将它的叶子剪掉，将它变成叶节点，决策树在验证集上的准确率不降低，则将它变成叶节点

- C4.5相比于ID3的优点是什么，C4.5有可能有什么缺点?

  ID3算法是决策树的一个经典的构造算法，但ID3算法也存在一些问题信息增益的计算依赖于特征数目较多的特征，而属性取值最多的属性并不一定最优。C4.5算法继承了ID3算法的优点，而且用信息增益率来选择属性，克服了用信息增益选择属性时偏向选择取值多的属性的不足。

  C4.5也存在一些缺点，在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。此外，C4.5只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

- 如何用决策树来选择特征?

  在构建决策树的时候有三种方法来选取特征，分别是ID3，C4.5和CART三种方法。但还有一种决策树的升级方法，就是随机森林算法。鉴于决策树容易过拟合的缺点，随机森林采用多个决策树的投票机制来改善决策树，每个树基于训练数据的随机样本，可以解决决策树常见的过拟合的问题。也可以使用决策树集合来计算每个特征的相对重要性。