实验

实验要求：

用 C4.5 算法对鸢尾花数据集进行分类。

C4.5 算法具体可网上查询具体算法进行学习。

数据：iris 数据集包含在 sklearn 库当中，具体在 sklearn\datasets\data 文件夹下，文件名为 iris.csv。也可自行网上下载。

环境：python 3

验收方式：实验报告

报告书写要求：

一、C4.5 算法

C4.5 是一系列用在机器学习和数据挖掘的分类问题中的算法。它的目标是**监督学习**：给定一个数据集，其中的每一个元组都能用一组属性值来描述，每一个元组属于一个互斥的类别中的某一类。C4.5 的目标是通过学习，找到一个从属性值到类别的映射关系，并且这个映射能用于对新的类别未知的实体进行分类。

C4.5 由 J.Ross Quinlan 在 ID3 的基础上提出的。ID3 算法用来构造决策树。决策树是一种类似流程图的树结构，其中每个内部节点（非树叶节点）表示在一个属性上的测试，每个分枝代表一个测试输出，而每个树叶节点存放一个类标号。一旦建立好了决策树，对于一个未给定类标号的元组，跟踪一条有根节点到叶节点的路径，该叶节点就存放着该元组的预测。决策树的优势在于不需要任何领域知识或参数设置，适合于探测性的知识发现。

C4.5 算法用**信息增益率**来选择属性。ID3 选择属性用的是子树的信息增益，这里可以用很多方法来定义信息，ID3 使用的是熵（entropy，熵是一种不纯度度量准则），也就是熵的变化值，而 C4.5 用的是信息增益率，公式：

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2(\frac{|D_j|}{|D|})$$

在决策树构造过程中进行剪枝，因为某些具有很少元素的结点可能会使构造的决策树过适应（Overfitting），如果不考虑这些结点可能会更好。

算法优点：产生的分类规则易于理解，准确率较高。

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。

二、程序实现

A. 数据处理

首先得到全部数据：

```
def load_preprocess_data():
    iris_data = []
    cnt = 0
    # 得到全部数据
    with open("iris.csv") as f:
        lines = f.readlines()
        for line in lines:
            if cnt == 0:
                cnt += 1
                continue
            tmp = line.strip().split(',')
            for i, e in enumerate(tmp):
                if i < 4:
                    tmp[i] = float(e)
            iris_data.append(tmp)
    print("数据集：\n", iris_data)
    iris_labels = ["花萼长度", "花萼宽度", "花瓣长度", "花瓣宽度"]
```

由于四种特征全为连续型数值，将其转换为离散型特征，找到分界点的阈值。Petal_width 的分界情况如图所示

```
categ_sepal_len = pd.cut(iris['sepal_len'],3)
catag_sepal_wid = pd.cut(iris['sepal_width'], 3)
catag_petal_len = pd.cut(iris['petal_len'], 3)
catag_petal_wid = pd.cut(iris['petal_width'], 3)
```

```
(0.9, 1.7]       54
(0.0976, 0.9]    50
(1.7, 2.5]       46
Name: petal_width, dtype: int64
```

图 1 数据集分界点

划分数据集，使得测试集为 20%，其余 80%为训练集

```
def get_train_test_dadaset(iris_data, split = 0.8):
    num = len(iris_data)
```

```
    train_data = []
    test_data = []
    for i in range(num):
        if i % 5 == 0:
            test_data.append(iris_data[i])
        else:
            train_data.append(iris_data[i])
    return train_data, test_data
```

B. 公式代码

因 C4.5 算法中使用到了较多如信息熵、信息增益率等公式，因此对于这些公式进行代码定义

```
# 计算数据集香农信息熵
def cal_shannon_entropy(dataset):
    num = len(dataset)
    label_dic = {}
    for row in dataset:
        if row[-1] not in label_dic.keys():
            label_dic[row[-1]] = 0
        label_dic[row[-1]] += 1
    entropy = 0
    for key in label_dic:
        p = label_dic[key] / float(num)
        entropy -= p * np.log2(p)
    return entropy
```

C. 选择最优特征

```
# c4.5 算法选择最优特征
def choose_feature_to_split(dataset, c45 = False):
    length = len(dataset)
    num_feature = len(dataset[0]) - 1      #特征个数
    base_entropy = cal_shannon_entropy(dataset)
    best_info_gain = 0
    best_info_gain_ratio = 0
    best_feature_index = -1
    for feature_index in range(num_feature):
        feat_list = [row[feature_index] for row in dataset]
        feat_dic = Counter(feat_list) #统计特征种类及出现次数
        cur_entropy = 0
        cur_feat_entropy = 0    # 数据集关于当前特征的信息熵, 用于 c4.5 算法计算
        for feat in feat_dic:
            sub_dataset = split_dataset(dataset, feature_index, feat)
            sub_entropy = cal_shannon_entropy(sub_dataset)
            cur_entropy += feat_dic[feat] / float(length) * sub_entropy
```

```
                cur_feat_entropy -= feat_dic[feat] / float(length) * np.log2(feat_dic[feat] /
float(length))
            # C4.5 算法计算
            gain_ration = (base_entropy - cur_entropy) / cur_feat_entropy
            if gain_ration > best_info_gain_ratio:
                best_info_gain_ratio = gain_ration
                best_feature_index = feature_index
    return best_feature_index
```

D. 构建决策树

```
# 深度优先构建决策树
def create_decision_tree(dataset, labels, c45):
    class_list = [row[-1] for row in dataset]
    # 如果数据集全部是同一个类别，直接返回该类别
    if len(class_list) == class_list.count(class_list[0]):
        return class_list[0]
    # 如果没有特征可以继续划分或者特征区分度为 0，返回出现次数最多的类别
    if len(dataset[0]) == 1 or choose_feature_to_split(dataset, c45) == -1:
        class_dic = Counter(class_list)
        return class_dic.most_common()[0][0]

    # 根据 c4.5 算法选择信息增益最高的特征作为当前分类特征
    best_feature_index = choose_feature_to_split(dataset, c45)
    best_feature_label = labels[best_feature_index]
    labels.remove(best_feature_label)
    # 以字典的形式构建决策树
    decision_tree = {best_feature_label : {}}
    feat_list = [row[best_feature_index] for row in dataset]
    feat_dic = Counter(feat_list)
    # 根据当前最优特征划分子数据集
    for feat in feat_dic:
        sub_dataset = split_dataset(dataset, best_feature_index, feat)
        sub_labels = labels[:]
        # 递归构建子数据集决策树
        decision_tree[best_feature_label][feat] = create_decision_tree(sub_dataset, sub_labels)
    return decision_tree
```

三、分类结果

分类结果如图所示，可以看到，在选取 150 组数据中 20%（即 30 组）数据作为测试集的情况下，有六组错误分类，准确率为 80%。

```
[1, 1, 1, 1, 'Iris-versicolor']  wrong answer: Iris-virginica
[1, 1, 1, 1, 'Iris-versicolor']  wrong answer: Iris-virginica
[1, 1, 1, 1, 'Iris-versicolor']  wrong answer: Iris-virginica
[1, 1, 1, 1, 'Iris-versicolor']  wrong answer: Iris-virginica
[1, 1, 1, 1, 'Iris-versicolor']  wrong answer: Iris-virginica
[1, 0, 1, 1, 'Iris-virginica']  wrong answer: Iris-versicolor

Accuracy : 80.00%
```

图 2 分类结果

但通过分析发现，构建的决策树是略有问题的，属于同一种的数据，却有分支。这里分析是因为鸢尾花数据集本来是连续性数据，这里**强行离散化**处理，并不能很好的进行区分，所以在这些分支，每一个分支下对应的数据均没有把数据完全分开，即未达到递归的第一个终止条件，达到了第二个条件，而且在每个分支中最大的都是同一种标签，就出现了这种情况。后期需要针对这种情况，进一步改进优化。

四、收获与体会

本次实验中，学习了使用 C4.5 算法对鸢尾花进行分类，C4.5 算法是基于机器学习中较常使用的决策树算法，进行特征点的选择。算法中涉及到信息熵、信息增益、信息增益率等信息论中的内容，算法解释性较好，对于分类的结果准确度也较好。

相较于上一次 k-nn 算法，此次实验对于代码编写、数学理解都有了一定的培养，也从两次对同一任务使用不同算法中，体会到了机器学习的乐趣。也从分类结果中，分析处理造成准确度不太理想的主要原因，也为后续的改进工作和日后的学习，打下基础。