《人工智能导论》

实验五：决策树 ID3 算法

学　号 22920212204396

姓　名 黄子安

2024 年 5 月 16 日

实验五：决策树 ID3 算法

229202212204396 黄子安

一、实验目的

编程实现决策树算法 ID3；理解算法原理。

二、实验内容

1.利用 taindata.txt 的数据（75*5，第 5 列为标签）进行训练，构造决策树；

2.利用构造好的决策树对 testdata.txt 的数据进行分类，并输出分类准确率。

三、实验过程

先编写一个决策树类，其中提供成员变量 min_samples_split 和 max_depth

- 前一个变量用于提前结束训练，当对应的结点中种类小于 min_samples_split 时就认为决策树的分裂可以结束，减少过拟合现象的发生；
- 第二个变量用于剪枝，当决策树的高度超过 max_depth 时直接结束训练，避免决策树过高导致效率降低，

之后定义熵的计算函数，使用 np.unique 对样本进行去重和排序，之后通过 mean 和逻辑相等获得对应的概率值，最后计算熵值，对应的公式为：

$$entropy = -\sum p_i log_2(p_i)$$

```
class DecisionTree:
    def __init__(self, min_samples_split=2, max_depth=None):
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def entropy(self, y):  # 计算熵
        classes = np.unique(y)
        entropy = 0
        for cls in classes:
            p_cls = np.mean(y == cls)
            entropy += -p_cls * np.log2(p_cls)
        return entropy
```

决策树类向外界提供 fit 方法用于决策树的构建与训练，具体的流程如下：

- 如果当前的结点中只含有一个类，则递归结束，构造对应的叶子节点
- 如果到达前面所说的剪枝条件也直接进行剪枝，结束递归调用，因为此时节点中可能有不止一个类别，因此选择对应出现概率最高的类
- 如果没有达到递归结束条件，则继续递归，先寻找最优的划分方案，使得划分后的信息增益达到最大，使用对应的指标和阈值进行节点的分裂，递归调用 fit 函数完成决策树的构造，这里使用字典来保存决策树，方便进行递归的存储

```
    def fit(self, X, y, depth=0):
        if (self.max_depth is not None and depth ⩾ self.max_depth) or \
                len(y) < self.min_samples_split or len(np.unique(y)) == 1:
            return {'class': np.argmax(np.bincount(y.astype(int)))}
        else:
            best_feature_idx, best_threshold = self.find_best_split(X, y)
            left_idxs = X[:, best_feature_idx] < best_threshold
            right_idxs = ~left_idxs
            tree = {
                'feature_idx': best_feature_idx,
                'threshold': best_threshold,
                'left': self.fit(X[left_idxs], y[left_idxs], depth + 1),
                'right': self.fit(X[right_idxs], y[right_idxs], depth + 1)
            }
            return tree
```

寻找最优划分的方法如下，对于当前节点，枚举所有的特征指标和所有的阈值，根据贪心思想，选出最大的信息增益，最后将划分的指标和划分的阈值返回

```
    def find_best_split(self, X, y):  # 找出最优划分
        best_gain = 0
        best_feature_idx = None
        best_threshold = None
        for feature_idx in range(X.shape[1]):
            thresholds = np.unique(X[:, feature_idx])
            for threshold in thresholds:
                gain = self.information_gain(X, y, feature_idx, threshold)
                if gain > best_gain:
                    best_gain = gain
                    best_feature_idx = feature_idx
                    best_threshold = threshold
        return best_feature_idx, best_threshold
```

计算信息增益的函数如下所示，给定当前的样本和划分阈值，会将节点分裂成两个子节点，先计算分裂之前父节点的熵值，之后计算分裂后两个子节点的熵值，同时对于子节点的熵值要**乘以一个权重**，否则在分裂的时候会导致决策树更容易选择具有大量不同值的属性，这样会导致过拟合和树过高，这里的权重使用子节点的比例即可，综上设将训练元组 D 按属性 A 进行划分，则 A 对 D 划分的期望信息增益即为

$$gain(A) = entropy(D) - entropy_A(D)$$

$$entropy_A(D) = \sum \frac{|D_i|}{|D|} entropy(D_i)$$

```
    def information_gain(self, X, y, feature_idx, threshold):  # 计算信息增益
        parent_entropy = self.entropy(y)
        left_idxs = X[:, feature_idx] < threshold
        right_idxs = ~left_idxs
        if np.sum(left_idxs) == 0 or np.sum(right_idxs) == 0:
            return 0
        left_entropy = self.entropy(y[left_idxs])
        right_entropy = self.entropy(y[right_idxs])
        child_entropy = np.mean(left_idxs) * left_entropy
                        + np.mean(right_idxs) * right_entropy
        return parent_entropy - child_entropy
```

至此完成了决策树的训练算法，之后给出对应的预测方法，类似于二叉搜索树，如果到达叶子节点则返回对应的标签，否则根据节点对应指标的值和阈值的关系进入左子树或者右子树进行递归即可

```
    def predict_sample(self, sample, tree):
        if 'class' in tree:
            return tree['class']
        else:
            if sample[tree['feature_idx']] < tree['threshold']:
                return self.predict_sample(sample, tree['left'])
            else:
                return self.predict_sample(sample, tree['right'])

    def predict(self, X, tree):
        return [self.predict_sample(sample, tree) for sample in X]
```

之后提供方法实现可视化决策树，训练完成后的决策树通过字典进行保存，因此根据当前节点是否为叶子节点进行递归即可，这里通过使用 graphviz 库来实现绘制决策树，graphviz 是一个开源的图形可视化软件，能够从简单的文本文件描述中生成复杂的图形和网络，此外为了作图更加美观，给相同种类的叶子节点打上相同的颜色

```python
def visualize_tree(tree, parent=None, graph=None, node_id=None, leaf_colors=None):
    colors = ['lightblue', 'lightgreen', 'lightcoral',
          'lightskyblue', 'lightpink', 'lightyellow']
    if leaf_colors is None:
        leaf_colors = {}
    if graph is None:
        graph = Digraph()
        graph.attr('node', shape='box')
    if node_id is None:
        node_id = '1'

    if 'class' in tree:  # Leaf node
        class_label = tree['class']
        if class_label not in leaf_colors:
            leaf_colors[class_label] = colors[len(leaf_colors) % len(colors)]
        graph.node(node_id, label=f'Class: {class_label}'
                   , color=leaf_colors[class_label], style='filled')
    else:  # Decision node
        feature_idx = tree['feature_idx']
        threshold = tree['threshold']
        graph.node(node_id, label=f'Feature {feature_idx} ⩽ {threshold:.2f}')

        left_child_id = f'{node_id}L'
        right_child_id = f'{node_id}R'

        graph.edge(node_id, left_child_id, label='Yes')
        graph.edge(node_id, right_child_id, label='No')

        visualize_tree(tree['left'], parent=node_id, graph=graph,
                       node_id=left_child_id, leaf_colors=leaf_colors)
        visualize_tree(tree['right'], parent=node_id, graph=graph,
                       node_id=right_child_id, leaf_colors=leaf_colors)

    if parent is None:
        return graph
```

最后编写数据读入方法和主方法，在主方法中先读入数据，再实例化一个决策树对象，进行训练后对测试集进行测试，使用 sklearn 库提供的方法计算对应的分类指标并打印输出

```
def read_data():
    data = np.genfromtxt('traindata.txt')
    X_train = data[:, :-1]
    y_train = data[:, -1]

    data = np.genfromtxt('testdata.txt')
    X_test = data[:, :-1]
    y_test = data[:, -1]
    return X_train, X_test, y_train, y_test


if __name__ == '__main__':
    X_train, X_test, y_train, y_test = read_data()
    dt = DecisionTree()
    tree = dt.fit(X_train, y_train)

    graph = visualize_tree(tree)
    graph.view()
    Source(graph.source)

    y_pred = dt.predict(X_test, tree)
    accuracy = accuracy_score(y_test, y_pred)
    print("Accuracy:", accuracy)
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
```

四、实验结果

经过训练后在测试集上的表现如下图所示，可以看到识别的准确率达到了93%，证明决策树分类效果不错

```
D:\anaconda3\python.exe D:\Desktop\learning\3.2\人工智能导论\lab\lab5\code.py
Accuracy: 0.9333333333333333
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98        25
         1.0       0.92      0.88      0.90        25
         2.0       0.92      0.92      0.92        25

    accuracy                           0.93        75
   macro avg       0.93      0.93      0.93        75
weighted avg       0.93      0.93      0.93        75


Process finished with exit code 0
```

同时，代码也生成了对应的可视化决策树如下所示：