决策树：从原理到应用

易凯

西安交通大学人工智能与机器人研究所

西安交通大学社会心理学研究所

西安交通大学软件学院

学号: 2151601053

班级：软件 53 班

邮箱：williamyi96@gmail.com

Abstract

这篇文章主要探讨的是决策树相关内容。我们首先回顾了决策树的基本概念，接着说明了此文章需要完成的任务，然后一方面基于 Scikit-learn 官网示例程序对其相关原理以及实现有了更进一步深入的认识，另外一方面分析了给定任务的基本思路。我们在试验中采用的是 CART 决策树，实验结果表明，该方法所生成的决策树具有一定的不稳定性，除了分析数据本身可能产生的过拟合问题，未来将会探讨研究更为稳定的决策树。

1 决策树基本概念

决策树 (Decision Tree) 是用于分类和回归的非参数监督学习方法，其目标是构建一个可解释的模型，通过学习从数据特征推断出对于事件发生满足条件的决策。

当前，常用的决策树算法主要有 ID3 (Iterative Dichotomiser 3) [1], C4.5 [2], CHi-squared Automatic Interaction Detector (CHAID) [3] 算法等。

与其他机器学习领域的小样本分类与回归任务相比，决策树具有如下优势 1：

Table 1: 与其他小样本分类与回归任务相比，决策树主要优势

优点	说明
可解释性	决策树易于理解与解释，树可以被可视化。
小样本学习	决策树使用统计测试的方式来验证模型的有效性，其说明了模型的可靠性。

Table 2: 与其他小样本分类与回归任务相比，决策树主要劣势

缺点	说明
缺乏归纳性	决策树学习过程中极有可能创建过于复杂的树，不能够对数据属性进行表征，缺乏概括性
不稳定性	训练数据的微小改变可能会导致生成完全不同的树。
决策的有穷依赖性	决策树不能够很好地学习诸如 XOR，奇偶校验等概念
非平衡性	如果某些类占主导地位，决策树学习者会创建偏向性树。
局部最优性	决策树只能基于启发式算法，由于该问题的最优化部分是 NP 完全的，因此只能保证局部最优

于此同时，处于决策树自身的特性，其具有主要劣势如表 2：

2 题目要求描述

使用决策树方法生成选定数据库文件中的图片所表示数据集的决策树，并对第 15 天的 play 结果进行预测。

此外，使用 Jupyter Noteboot 以及 Scikit-learn IRIS 数据库对相关数据操作进行测试。

扩展之后的题目要求分为两个部分，一部分是根据 scikit-learn 相关语法在 Jupyter Notebook 环境之下，对相关方法进行进一步更为深入的理解与掌握。另外就是面向题目中给定的数据集，进行第 15 天之后结果的测试。

使用到的数据集为：

D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast		High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No

D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

3 分析思路

对于第一个任务，按照官方的指导，逐条完成给定的内容即可，由于主要是为了熟悉相关语法，因此掌握以及实践起来难度不大。对于第二个任务，其需要基于 scikit-learn 对决策树基本概念以及实现量化建模以及应用，首先需要的是将文本数据编码，接着将编码之后的文本数据导入进来，使用 scikit-learn 自带决策树函数完成 CART 决策树 [4] 生成, 然后使用 pydotplus 或者 graphviz 进行决策树的拓扑结构生成，最后基于决策树对新输入数据进行决策判定。

4 Scikit-learn 决策树测试

按照 scikit-learn 官网上的基本介绍，完成的测试如下图所示。

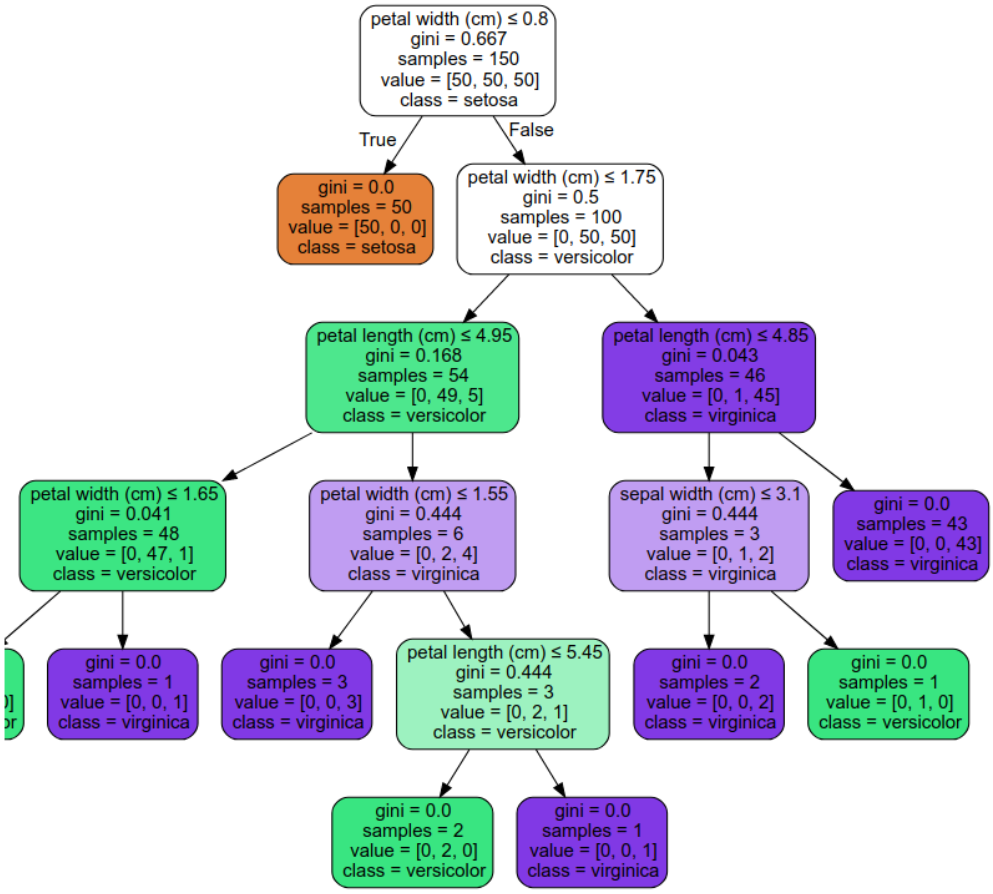
In [1]:
```
from sklearn import tree
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
import graphviz
```

In [2]:
```
iris = load_iris() # load the IRIS data
clf = tree.DecisionTreeClassifier()
clf = clf.fit(iris.data, iris.target)
```

In [3]:
```
dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=iris.feature_names,
                                class_names=iris.target_names,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
```

In [4]:
```
graph
```

Out[4]:




```
In [7]: clf.predict(iris.data[:1, :])
Out[7]: array([0])

In [9]: clf.predict_proba(iris.data[:1, :])
Out[9]: array([[ 1.,  0.,  0.]])
```

5 代码实现

该部分主要内容为介绍给定数据集上的决策实现。

```
from sklearn import tree
from sklearn.preprocessing import LabelEncoder
from sklearn.externals.six import StringIO
import pandas as pd
# from sklearn.datasets import load_iris
import numpy as np
```

```
# import matplotlib.pyplot as plt
import pydotplus
import graphviz
import cv2

# iris = load_iris() # load the IRIS data
# clf = tree.DecisionTreeClassifier()
# clf = clf.fit(iris.data, iris.target)

# dot_data = tree.export_graphviz(clf, out_file=None,
#                                 feature_names=iris.feature_names,
#                                 class_names=iris.target_names,
#                                 filled=True, rounded=True,
#                                 special_characters=True)
# graph = graphviz.Source(dot_data)
# print(graph)

def decision_tree():
        with open('decision.txt', 'r') as fr:
                decision = [inst.strip().split('\t') for inst in fr.readlines()]
        target = []
        for each in decision:
                target.append(each[-1])
        # print(target)  # print the decision

        decision_list = []
        decision_dict = {}
        labels = ['Day', 'Weather', 'Humidity', 'Wind']

        for each_label in labels:
                for each in decision:
                        decision_list.append(each[labels.index(each_label)])
```

```
                decision_dict[each_label] = decision_list
                decision_list = []
    # print(decision_dict) # print the data according to every colon
        decision_pd = pd.DataFrame(decision_dict)
        # print(decision_pd) # display all the data in a sequence (start from 0)
        labelencoder = LabelEncoder()
        for col in decision_pd.columns:
                decision_pd[col] = labelencoder.fit_transform(decision_pd[col])
        # print(decision_pd)
    # print(decision_pd.keys())

        clf = tree.DecisionTreeClassifier(max_depth = 8)
        clf = clf.fit(decision_pd.values.tolist(), target)

        dot_data = StringIO()
        tree.export_graphviz(clf, out_file = dot_data,
                                        feature_names = decision_pd.keys(),
                                        class_names = clf.classes_,
                                        filled=True, rounded=True,
                                        special_characters=True)
        graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    # print()
    # graph.render("Decision Tree")
        graph.write_pdf("decision_tree.pdf")

        # print(clf.predict([['D15','Rain','High','Weak']]))
        print(clf.predict([[16,0,1,1]]))

if __name__ == '__main__':
    decision_tree()
```

6 实验结果

通过上述代码运行得到的决策树如图 1 所示：

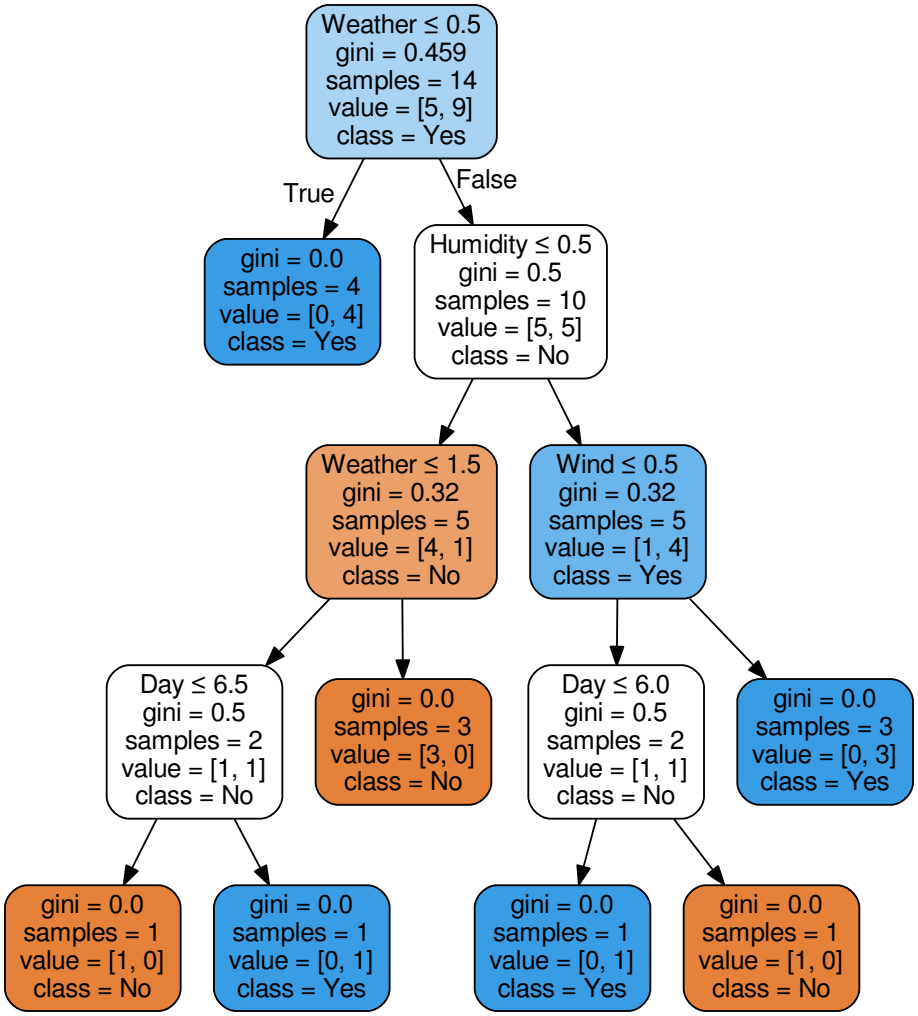


Figure 1: 通过实验数据采用 CART 决策树生成的决策树模型

该实验在不同的测试周期有着不一样的测试结果，有‘yes’以及‘No’两种可能输出，分析实验结果表明，很可能是出于使用 CART 生成决策树的不稳定性造成的。

7 总结与未来工作

该文章对于决策树的基本概念有了细致的回顾与总结，同时结合 Scikit learn 官方示例对决策树的概念及其使用有了进一步的认识，此外，分析了决策树潜在的问题，并提出了潜在的解决方案。未来的工作将主要集中在如何设计较为简单，但是可以有效防止过拟合的决策树。

致谢

感谢赵老师的指导以及点拨，感谢帮助过我的师兄师姐，感谢开源社区提供无私的答疑解惑的平台。

References

[1] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[2] J. Ross Quinlan. C4.5: programs for machine learning. 1, 1993.

[3] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Journal of the Royal Statistical Society*, 29(2):119–127, 1980.

[4] Leszek Rutkowski, Maciej Jaworski, Lena Pietruczuk, and Piotr Duda. The cart decision tree for mining data streams. *Information Sciences*, 266(5):1–15, 2014.