

DECISION TREES

赖昱行

Yuhang Lai

北京理工大学

Beijing Institute of Technology

日期: 2022 年 5 月 14 日

摘 要

本文为机器学习初步课程大作业的作业报告, 作业使用 **Python** 语言实现。报告解释了如何设计和实现 `inspection` 和 `decisionTree` 两个分类器, 并介绍了实验结果, 对结果进行了分析。

1 inspection and decisionTree

1.1 结构设计

`inspection` 作为第一个分类器, 实质上是简化版的 `decisionTree`。根据作业要求, 使用了 majority vote 的分类方法, 仅仅构建了决策树的第一层, 即不考虑任何特征属性, 直接根据类别的数量信息进行建树。

`decisionTree` 作为第二个分类器, 是一棵以特征属性的 mutual information 作为分裂参考的二分类决策树。同时, 作业要求可以对决策树的深度给出限制。本次实验设计了一个决策树类, 可以在给定的数据上从根节点出发, 以递归的形式进行分类建树, 在递归过程中对深度限制, 决策方法进行实现。

1.2 代码实现

`inspection` 简单地设计了一个函数 `inspect`。函数根据命令行参数确定输入输出路径, 首先对输入的数据进行格式处理, 而后简单的统计各个类别的数量, 并计算 entropy 以及 error rate。计算 entropy 的公式如下

$$H(X) = - \sum_X P(X=x) \log P(X=x) \quad (1)$$

`decisionTree` 设计了一个类 `MyDecisionTree`。此类对命令行参数进行了保存, 每次训练或者测试时可将数据存入变量 `data` 中, 建树结果存入 `tree` 中, 然后即可在训练好的决策树上进行测试, 将结果写入对应路径的文件中。

`build` 函数首先初始化了决策树的树根, 然后调用了 `treeSplit` 函数对树进行递归式的分裂。`treeSplit` 函数作为代码的核心部分, 通过在每个节点中记录 `data` 中的 index, 即 `now_data_idx` 来实现对应数据的保存, 而后又使用类似的思想, 通过 `ignore_feature_idx` 记录当前节点到根路径上的 feature, 以此可实现后续简单的可用 feature 遍历。深度的控制十分简单, 这里不再赘述。

在 *treeSplit* 函数中，给树中的节点划分了三个属性 *Type*, *Child*, *Data*，使用 *dict* 实现。在 *Type* 中，存放有当前节点特征属性的编号；在 *Child* 中，有另一个 *dict*，其中存放有每一个特征对应的子节点；在 *Data* 中，存放有 *now_data_idx*。在 *treeSplit* 中的另一个重要函数 *findMost* 用来找到确定当前节点的分裂决策，使用了 *mutual information* 作为决策标准，公式如下

$$MI(X, Y) = H(Y) - \sum_X P(X = x) H(Y|X = x) \quad (2)$$

建树完成后，使用 *printTree* 函数按要求打印决策树的形态。以上过程都在 *train* 函数中得到集成，使用 *predict* 函数进行结果预测。同理，在测试时，*test* 函数也使用了 *predict* 进行了预测。*train* 和 *test* 的标签和评测指标均按要求输出到相应的文件中。

2 Experimentation

2.1 实验结果

针对 *inspection*，执行如下命令

```
python3 inspection.py small_train.tsv small_inspect.txt
```

得到与样例一致的结果

```
entropy: 0.996316519559
error: 0.464285714286
```

对于 *decisionTree*，首先对样例进行验证，即在 *small* 数据集上构建一棵深度为 3 的决策树。执行如下命令

```
python3 decisionTree.py small_train.tsv small_test.tsv 3 small_3_train.labels small_3_test.labels small_3_metrics.txt
```

与 *handout* 中的标准答案对比后，结果一致。

随后在各数据集上进行测试，*decisionTree* 设定深度为 3,10，结果如下表

inspection Results		
Dataset	Entropy	Error Rate
small_train.tsv	0.996316519558962	0.4642857142857143
small_test.tsv	0.996316519558962	0.4642857142857143
politicians_train.tsv	0.990589428653754	0.4429530201342282
politicians_test.tsv	0.999895287418619	0.4939759036144578
education_train.tsv	0.909736122531166	0.3250000000000000
education_test.tsv	0.893173458377857	0.3100000000000001
mushroom_train.tsv	0.962614705998252	0.3866666666666667
mushroom_test.tsv	0.809031792220290	0.2485875706214690

表 1: Results of *inspection* running on different datasets

decisionTree Results			
Dataset	Depth	Train ER	Test ER
small	3	0.170000	0.205000
small	10	0.000000	0.000000
politicians	3	0.114094	0.168675
politicians	10	0.067114	0.204819
education	3	0.170000	0.205000
education	10	0.000000	0.000000
mushroom	3	0.022667	0.020716
mushroom	10	0.000000	0.020716

表 2: Results of decisionTree running on different datasets

对于命令

```
python3 decisionTree.py small_train.tsv small_test.tsv 3 train_out test_out metrics_out
```

程序打印决策树形态的结果如下

```
[15 democrat/13 republican]
| Anti_satellite_test_ban = n: [2 democrat/12 republican]
| | Export_south_africa = n: [0 democrat/5 republican]
| | Export_south_africa = y: [2 democrat/7 republican]
| Anti_satellite_test_ban = y: [13 democrat/1 republican]
| | Export_south_africa = n: [0 democrat/1 republican]
| | Export_south_africa = y: [13 democrat/0 republican]
```

2.2 结果分析

从实验结果可以看出，由于 decisionTree 对 inspection 的基础根节点进行了进一步的分裂，在各数据集上都取得了更优的 error rate。而对于 decisionTree 而言，决策树深度的增加稳定地减少了训练集上的 error rate，并在一半测试集上同样取得了更优的效果。虽然在 education 和 mushroom 数据集上没有观察到 error rate 的下降，推测出现了对训练集的过拟合，但是这也符合我们的预期。事实上，本次实验在 MyDecisionTree 类中预留了函数 prone 作为剪枝的接口，后续可进行相应扩展，优化决策树的分裂效果，缓解过拟合现象。

3 结语

本次实验使用 Python 语言手动实现了决策树，对决策树的实现和运行原理有了更加深入的理解。实验相关代码已上传至 Github¹。

¹[halfrot/naive-decision-tree](https://github.com/halfrot/naive-decision-tree)