**机器学习导论**

**编程作业：决策树**

姓名：陶威

班级：电信1604班

学号：U201613331

指导老师：王邦

# 任务一：使用决策树预测隐形眼镜类型

## 实验原理：

根据最大香农信息熵增益从训练数据中选择特征，递归地构建决策树，直至达到递归终止条件。

## 编程思路：

1. 构建DecisionTree的类，定义好所需要的api和接口
2. 在类里实现计算香农信息熵，选择最大增益等静态方法
3. 实现训练、预测和计算准确率等具体方法

## 实现源码：

*import* math  
*import* numpy *as* np  
*from* collections *import* Counter  
*from* typing *import* List  
  
  
*class* DecisionTree:  
 \_\_X\_train: np.array = *None* \_\_y\_train: np.array = *None* \_\_decision\_tree: dict = *None  
  
 def* fit(*self*, data: list):  
 dataset = np.array(data)  
 *self*.\_\_X\_train = dataset[:, :-1]  
 *self*.\_\_y\_train = dataset[:, -1]  
 *return self* @staticmethod  
 *def* \_\_calc\_entropy(X\_train: np.array, y\_train: np.array) -> float:  
 label\_counts = Counter(y\_train)  
 *# 计算信息熵* entropy = 0.0  
 *for* key *in* label\_counts:  
 prob = float(label\_counts[key]) / len(X\_train)  
 entropy -= prob \* math.log(prob, 2)  
 *return* entropy  
  
 @staticmethod  
 *def* \_\_split\_dataset(X\_train: np.array, y\_train: np.array, index: int, value: any) -> (np.array, np.array):  
 new\_index = [row *for* row, data *in* enumerate(X\_train) *for* i, v *in* enumerate(data) *if* i == index *and* v == value]  
 new\_X\_train = np.array([data *for* i, data *in* enumerate(X\_train) *if* i *in* new\_index])  
 new\_y\_train = np.array([data *for* i, data *in* enumerate(y\_train) *if* i *in* new\_index])  
 *return* new\_X\_train, new\_y\_train  
  
 *def* \_\_choose\_best\_feature(*self*, X\_train: np.array, y\_train: np.array) -> int:  
 features\_num = X\_train.shape[1]  
 base\_entropy = *self*.\_\_calc\_entropy(X\_train, y\_train)  
 *# 最优的信息增益值, 和最优的特征的编号* best\_gain, best\_feature = 0.0, -1  
 *# 计算按照各个特征分类的信息熵  
 for* i *in* range(features\_num):  
 feature\_list = set(X\_train[:, i])  
 new\_entropy = 0.0  
 *for* feature *in* feature\_list:  
 sub\_X\_train, sub\_y\_train = *self*.\_\_split\_dataset(X\_train, y\_train, i, feature)  
 prob = len(sub\_X\_train) / float(len(X\_train))  
 new\_entropy += prob \* *self*.\_\_calc\_entropy(sub\_X\_train, sub\_y\_train)  
 info\_gain = base\_entropy - new\_entropy  
 *if* info\_gain > best\_gain:  
 best\_gain = info\_gain  
 best\_feature = i  
 *return* best\_feature  
  
 *def* \_\_predict(*self*, tree: dict, test\_data: list, labels: List[str]) -> any:  
 root = list(tree.keys())[0]  
 value = tree[root]  
 root\_index = labels.index(root)  
 key = test\_data[root\_index]  
 feat\_value = value[key]  
 *# 判断分支是否结束  
 if* isinstance(feat\_value, dict):  
 class\_label = *self*.\_\_predict(feat\_value, test\_data, labels)  
 *else*:  
 class\_label = feat\_value  
 *return* class\_label  
  
 *def* predict(*self*, test\_data: list, labels: list) -> List[str]:  
 *if not self*.\_\_decision\_tree:  
 *self*.build\_tree(labels)  
 *return* [*self*.\_\_predict(*self*.\_\_decision\_tree, data, labels) *for* data *in* test\_data]  
  
 @staticmethod  
 *def* score(y\_predict: list, y\_true: list) -> float:  
 *return* sum(np.array(y\_predict) == np.array(y\_true)) / len(y\_true)  
  
 *def* \_\_build\_tree(*self*, X\_train: np.array, y\_train: np.array, labels: List[str]) -> dict *or* str:  
 *if* Counter(y\_train)[y\_train[0]] == len(y\_train):  
 *return* y\_train[0]  
 *if* len(X\_train[0]) == 1:  
 major\_label = Counter(y\_train).most\_common(1)[0]  
 *return* major\_label  
  
 best\_feat = *self*.\_\_choose\_best\_feature(X\_train, y\_train)  
 best\_feat\_label = labels[best\_feat]  
 tree = {best\_feat\_label: {}}  
 feature\_list = set(X\_train[:, best\_feat])  
 *for* value *in* feature\_list:  
 *# 遍历当前选择特征包含的所有属性* tree[best\_feat\_label][value] = *self*.\_\_build\_tree(  
 \**self*.\_\_split\_dataset(X\_train, y\_train, best\_feat, value),  
 labels  
 )  
 *return* tree  
  
 *def* build\_tree(*self*, labels: List[str]) -> dict:  
 *self*.\_\_decision\_tree = *self*.\_\_build\_tree(*self*.\_\_X\_train, *self*.\_\_y\_train, labels)  
 *return self*.\_\_decision\_tree  
  
 *def* \_\_draw\_tree(*self*, tree: dict, depth: int) -> *None*:

# 根据层级关系简单地打印树  
 *if not* tree:  
 *return  
 if* isinstance(tree, str):  
 print(depth, '-' \* 2 \* depth, tree)  
 *return  
 for* key *in* tree:  
 print(depth, '-' \* 2 \* depth, key)  
 *self*.\_\_draw\_tree(tree[key], depth + 1)  
  
 *def* draw\_tree(*self*, tree):  
 *return self*.\_\_draw\_tree(tree, 0)

# 任务二：根据用户采集的WiFi信息采用决策树预测用户所在房间

## 实验原理：

根据BSSID的指纹将训练数据进行分组，在每一个分组中构建BSSID的特征向量，最后根据所得到的特征集来构建决策树进行训练，再来预测测试数据并和结果进行匹配计算准确率。

## 编程思路：

1. 导入训练集和测试集数据，准备进行数据处理和提取

*with* open('data/TrainDT.csv', 'r', encoding='iso-8859-1') *as* f:  
 train\_data = [line.strip().split(',') *for* line *in* f.readlines()]  
  
*with* open('data/TestDT.csv', 'r', encoding='iso-8859-1') *as* f:  
 test\_data = [line.strip().split(',') *for* line *in* f.readlines()]

1. 求出训练集中所有数据的BSSID并集，算出其长度为特征向量的长度

*# 求所有训练数据中BSSID的并集*bssids = list(set(np.array(train\_data)[:, 0]))  
feature\_num = len(bssids)

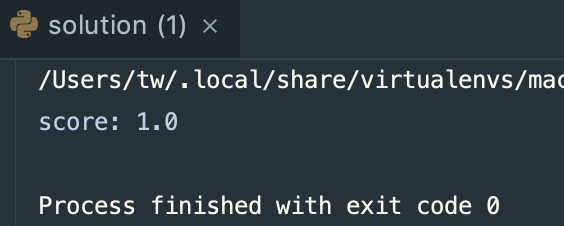
1. 将训练数据集按照finLabel进行分组，对每一个分组中的值构建一个特征向量，并且同时保存其对应的RoomLabel

*def* generate\_feature(data):  
 labels = data[0]  
 dataset = data[1:]  
 X = []  
 y = []  
 *# 按指纹分组* train\_data\_group = groupby(dataset, *lambda* x: x[-1])  
 *for* fin, group *in* train\_data\_group:  
 feature = [0] \* feature\_num  
 room\_label = 0  
 group = list(group)  
 *for* line *in* group:  
 *# 再做一次判断是因为测试数据集中可能出现训练数据集中没有的BSSID  
 if* line[0] *in* bssids:  
 feature[bssids.index(line[0])] = 1  
 room\_label = int(line[2])  
 X.append(feature)  
 y.append(room\_label)  
 *return* X, y, labels

1. 使用决策树训练数据，最后用测试集验证并计算准确度

X\_train, y\_train, labels = generate\_feature(train\_data)  
X\_test, y\_test, labels = generate\_feature(test\_data)  
dt\_clf = DecisionTree()  
dt\_clf.fit(X\_train, y\_train)  
print('score:', dt\_clf.score(X\_test, y\_test))

1. 测试结果截图：



# 任务三：IMDB数据集电影评测分类（二分类问题）

## 实验原理：

因每条数据维数不统一，需要进行one-hot编码来预处理数据。又因数据量较大，特征较多需要调试合理的参数来获得更好的准确率。可以设置一定范围的参数来不断尝试来获得最好的准确率，也可以采用一些库函数的网格搜索等方法。

## 编程思路：

1. 先读取数据并进行one-hot编码来预处理数据，这里的one-hot编码原理是以词库大小建立向量，每条数据中评论出现的位置标记为真值

*def* one\_hot\_encoder(feature\_num, data):  
 feature = np.zeros(feature\_num, dtype=np.int)  
 *for* num *in* data:  
 feature[int(num)] = 1  
 *return* feature

1. 将处理好的数据喂给决策树算法进行训练，同时调试相关参数达到准确度和成本的平衡
2. 读取测试集并计算相应的预测结果