机器学习第十一次作业

4.9 试将 4.4.2 节对缺失值的处理机制推广到基尼指数的计算中去.

$$Gini(D) = 1 - \Sigma_{k=1}^{|y|}\tilde{p}_k^2$$
$$Gini\_index(D,a) = \Sigma_{v=1}^{V}\tilde{r}_v Gini(\tilde{D}^v)$$

4.10 从网上下载或自己编程实现任意一种多变量决策树算法，并观察其在西瓜数据集 3.0 上产生的结果

1. 模块调用

```
import pandas as pd
import numpy as np
import treePlot


from sklearn.model_selection import train_test_split

from functools import reduce

from sklearn.utils.multiclass import type_of_target

# 使用LDA模型作为结点上的线性分类器
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

2. 数据处理
   - 多变量决策树采用表3.0α的数据（剔除所有离散属性）

```
# 载入数据
melon_dataset = pd.read_excel('C:/Users/mi/Desktop/DecisionTree/melon.xlsx')

# 划分数据集
#X = melon_dataset.iloc[:,1:-1]
# 西瓜数据集3.0a(忽略离散属性)
X = melon_dataset.loc[:,['密度', '含糖率']]
y = melon_dataset.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.4,stratify
= y, random_state = 5)
```

3. 结点类的定义

```
# 结点类
class Node(object):
    def __init__(self):
        # 属性名称
        self.feature_name = None
        # 属性编号(降低每个结点所占内存)
        self.feature_index = None
        # 子树集合 (dict: {featuretype: subtree})
        self.subtree = {}
        # 正例数占比
        self.impurity = None
```

```
        # 属性是否为连续变量
        self.is_continuous = False
        # 若为连续变量，定义临界值
        self.split_value = None
        # 是否为叶结点
        self.is_leaf = False
        # （作为叶结点）结点的类型
        self.leaf_class = None
        #  当前根节点对应决策树的叶子数
        self.leaf_num = None
        # 结点深度初始为-1
        self.high = -1
        # 分类器准确率
        self.accuracy = None
        # 结点是否为多变量
        self.is_multivariate = None
        # 决策边界法向量
        self.coef = None
        # 决策边界截距项
        self.intercept = None
```

4. 多变量决策树类的定义

```
# 决策树类
class Multivariate_Decision_Tree(object):
    # 不处理缺失值
    # 支持连续值情形
    # 采用信息增益作为划分依据
    def __init__(self, criterion = 'info_gain', pruning = None, multivariate = 
True):
        #: param criterion: 划分方式选择，目前仅支持‘info_gain’信息增益
        #: param pruning: 是否剪枝，可选择‘pre_pruning’ 与 ‘post_pruning’
        # 检验参数合法性
        assert criterion in ('gini_index','info_gain','gain_ratio')
        assert pruning in (None, 'pre_pruning','post_pruning')
        self.criterion = criterion
        self.pruning = pruning
        # 判断是否为多变量决策树
        self.multivariate = multivariate
```

5. 多变量决策树方法定义

```
# 训练接口
    def fit(self, X_train, y_train, X_valid = None, y_valid = None):
        #: param X_train: DataFrame类型数据 特征集合
        #: param y_train: DataFrame类型数据 分类标签
        #: param X_valid: DataFrame类型数据 剪枝特征集合
        #: param y_train: DataFrame类型数据 剪枝分类标签

        # 选择剪枝却未传入验证集
        if self.pruning is not None and (X_valid is None or y_valid is None):
            raise Exception('Please input validation data for pruning')

        # 输入验证集
        if X_valid is not None:
            pass
```

```
        # 存储特征名称
        self.columns = list(X_train.columns)

        # 建立决策树
        self.tree = self.generate_tree(X_train,y_train)

        return self
```

- 多变量决策树建树算法

```
    def generate_tree(self, X, y):
        #: param X: DataFrame类型训练数据 特征集合
        #: param y: DataFrame类型训练数据 分类标签
        # 初始化根节点
        my_tree = Node()
        my_tree.leaf_num = 0

        ####################### 递归终止条件
###################################
        # 样本全属于同一类别
        if y.nunique() == 1:
            # 将node标记为该类别的叶结点
            my_tree.is_leaf = True
            my_tree.leaf_class = y.values[0]
            # 根节点深度为0
            my_tree.high = 0
            # 根节点编号为1
            my_tree.leaf_num += 1
            return my_tree

        # 属性集为空或样本在属性上取值相同
        if X.empty or reduce(lambda x,y: x and y,(X.nunique().values ==
[1]*X.nunique().size)):
            # 标记为叶节点
            my_tree.is_leaf = True
            # 将样本中最多的类作为结点的类
            #my_tree.leaf_class = pd.value_counts(y).index[0]
            my_tree.high = 0
            my_tree.leaf_num += 1
            return my_tree



 #######################################################################

        # 从属性集中选择最优划分属性和对应分化指标
        # 多变量决策树要求在所有属性上进行划分
        # 返回结果  1. 多变量：best_feature_name, best_accuracy
        #            2. 单变量：best_feature_name, best_impurity(list)
        best_feature_name, best_accuracy, result, coef, intercept =
self.choose_best_multivariate_feature(X,y)

        # 根节点命名
        my_tree.is_multivariate = True
        my_tree.coef = coef
        my_tree.intercept = intercept
```

```
        my_tree.feature_name = best_feature_name
        my_tree.feature_accuracy = best_accuracy

        sub_X_pos = X[result == '是']
        sub_y_pos = y[result == '是']

        sub_X_neg = X[result == '否']
        sub_y_neg = y[result == '否']

        max_high = -1

        # 生成左子树
        my_tree.subtree['是'] = self.generate_tree(sub_X_pos, sub_y_pos)

        if my_tree.subtree['是'].high > max_high:
                    max_high = my_tree.subtree['是'].high
        my_tree.leaf_num += my_tree.subtree['是'].leaf_num

        # 生成右子树
        my_tree.subtree['否'] = self.generate_tree(sub_X_neg, sub_y_neg)

        if my_tree.subtree['否'].high > max_high:
                    max_high = my_tree.subtree['否'].high

        my_tree.leaf_num += my_tree.subtree['否'].leaf_num
        my_tree.high = max_high + 1

        return my_tree
```

- 线性分类器训练算法

在中间结点上利用LDA分类器划分训练数据，将LDA的决策面数据储存在结点中，按照递归的方式自上而下地生成新的子树。

```
    def choose_best_multivariate_feature(self,X,y):
        #: param X: DataFrame类型训练数据 特征集合(连续型)
        #: param y: DataFrame类型训练数据 分类标签
        #: return: [best_feature_name, best_info_gain]
        features = X.columns
        best_feature_name = None
        best_accuracy = None
        coef = None
        intercept = None
        result = None
        clf = None
        # 查找当前变量形成的决策边界
        clf = LDA()
        clf.fit(X,y)
        best_feature_name = '{0} * {1} + {2} * {3} >= {4}
?'.format(features[0], clf.coef_[0][0], features[1],clf.coef_[0][1], -
clf.intercept_[0])
        best_accuracy = clf.score(X,y)
        coef = clf.coef_[0]
        intercept = -clf.intercept_[0]
        # 分类结果
        result = clf.predict(X)
```

```
        return best_feature_name, best_accuracy, result, coef, intercept
```

- 预测

```
    def predict(self, X):
        #: param X : DataFrame类型测试数据
        #: return 若测试数据只有1条，返回值
        #         若有多条，返回向量
        # 检查实例中是否存在tree属性(是否已经拟合训练集数据)
        if not hasattr(self, "tree"):
            raise Exception('Please fit the data to generate a tree')

        if X.ndim == 1:
            return self.predict_single(X)
        else:
            return X.apply(self.predict_single, axis = 1)

    def predict_single(self, x, subtree = None):
        # 预测单一样例
        #:param x: 单一样例
        #:subtree 子树(预测起点的根节点)
        #:return

        # 默认从整棵树的根节点找起
        if subtree is None:
            subtree = self.tree

        # 子树为叶结点，返回叶结点类型作为预测结果
        if subtree.is_leaf:
            return subtree.leaf_class

        # 子树为多变量决策树
        if subtree.is_multivariate:
            if np.dot(x, subtree.coeff) >= subtree.intercept:
                return self.predict_single(x, subtree.subtree['是'])
            else:
                return self.predict_single(x, subtree.subtree['否'])
```

6. 主函数部分与训练结果

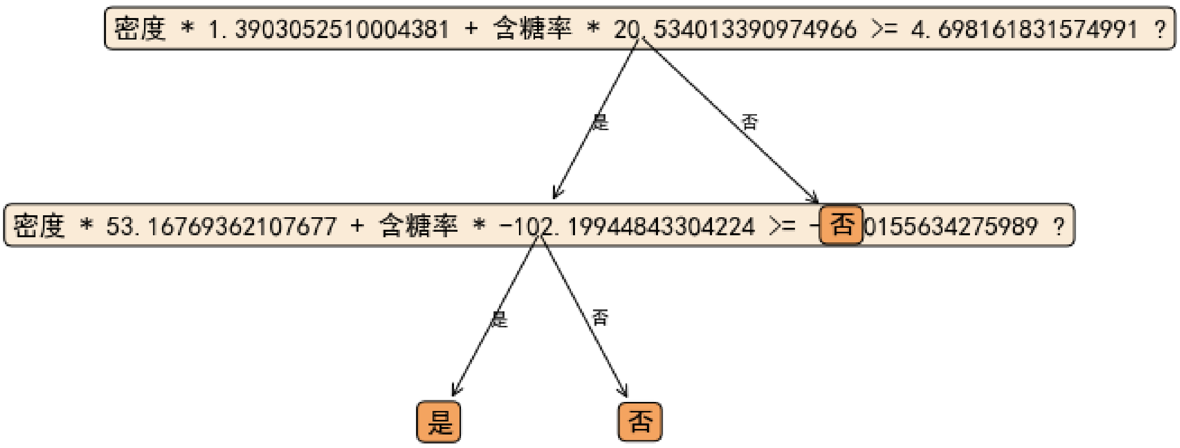
```
# 训练集
print('训练集：')
print(X_train)

# 验证集
print('验证集：')
print(X_test)


Mtree = Multivariate_Decision_Tree()
Mtree.fit(X_train, y_train)

treePlot.create_plot(Mtree.tree)
```

训练集：

	密度	含糖率
1	0.774	0.376
16	0.719	0.103
7	0.437	0.211
5	0.403	0.237
9	0.243	0.267
15	0.593	0.042
10	0.245	0.057
4	0.556	0.215
2	0.634	0.264
12	0.639	0.161

验证集：

	密度	含糖率
8	0.666	0.091
11	0.343	0.099
13	0.657	0.198
3	0.608	0.318
14	0.360	0.370
0	0.697	0.460
6	0.481	0.149



7. 实验结论
   - 多变量决策树对训练数据敏感
   - LDA作为结点分类器非常不好
     1. 深结点训练样例少，有可能出现三个样本在特征空间共线的情形（两个反例夹着一个正例），导致分类器将三个样例全部判为正例或反例。需要设定特殊的递归终止条件，否则无法停止递归。相比之下，SVM方法的决策边界更灵活，被多数研究采用。
     2. LDA在求解过程中需要求数据的逆矩阵，会出现variables are collinear的报错