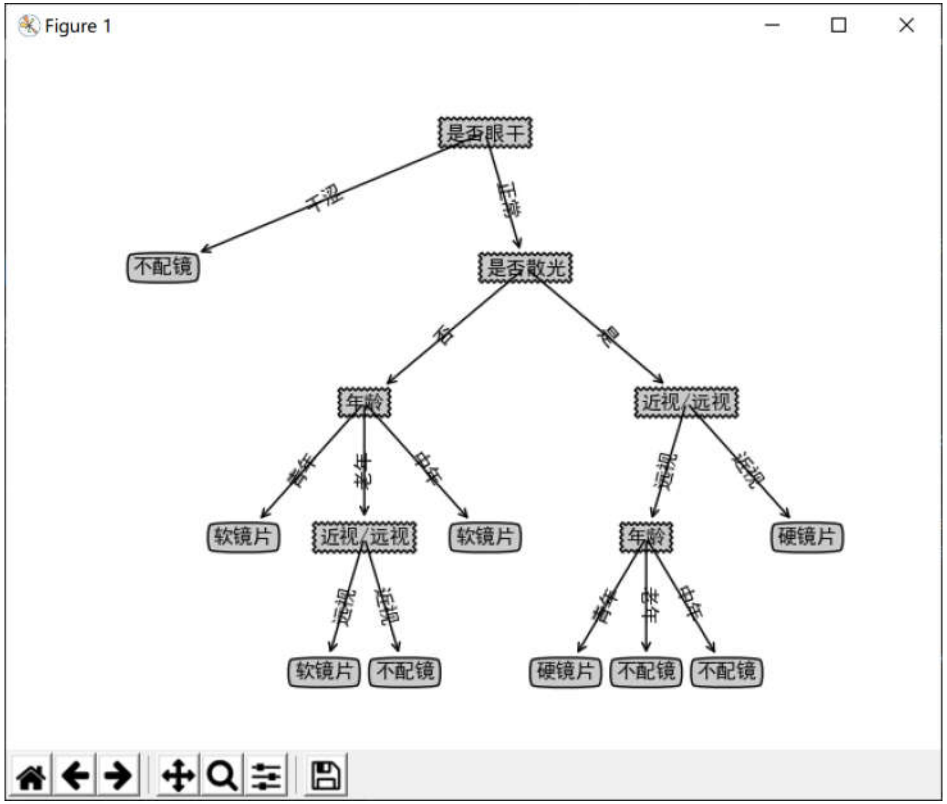
Python编程与人工智能实践

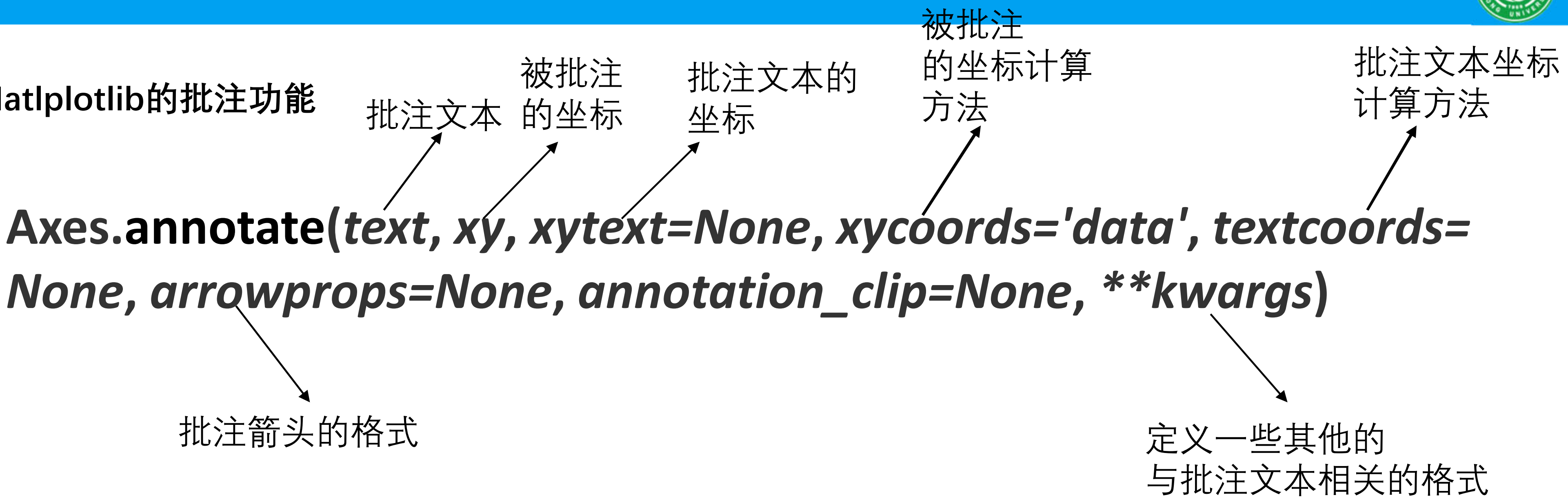


利用matplotlib 绘制决策树

于泓
鲁东大学
信息与电气工程学院

2019.11.13

Matlplotlib的批注功能



批注文本

被批注的坐标

批注文本的坐标

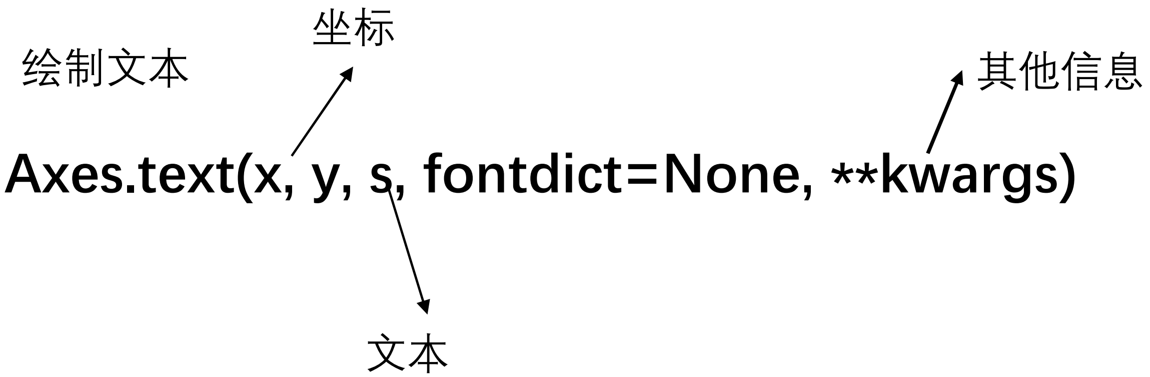
被批注的坐标计算方法

批注文本坐标计算方法

```
Axes.annotate(text, xy, xytext=None, xycoords='data', textcoords=None, arrowprops=None, annotation_clip=None, **kwargs)
```

批注箭头的格式

定义一些其他的与批注文本相关的格式



绘制文本

坐标

其他信息

```
Axes.text(x, y, s, fontdict=None, **kwargs)
```

文本

本任务中的应用

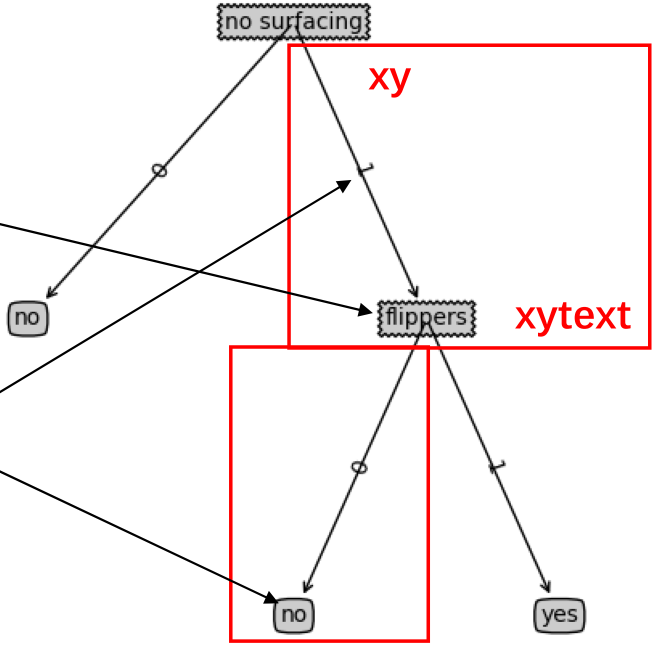
```
ax.annotate(nodeTxt, xy=parentPt,  xycoords='axes fraction',
                        xytext=centerPt, textcoords='axes fraction',
                        va="center", ha="center", bbox=nodeType,
                        arrowprops=arrow_args )
```

先画批注
再在箭头上
标注文字

```
decisionNode = dict(boxstyle="sawtooth", fc="0.8")

leafNode = dict(boxstyle="round4", fc="0.8")

arrow_args = dict(arrowstyle="<-")
```

xy -> xytext

```
ax.text(xMid, yMid, txtString, va="center", ha="center",
rotation=theta)
```



坐标计算方法:

每画一个 decisionNode

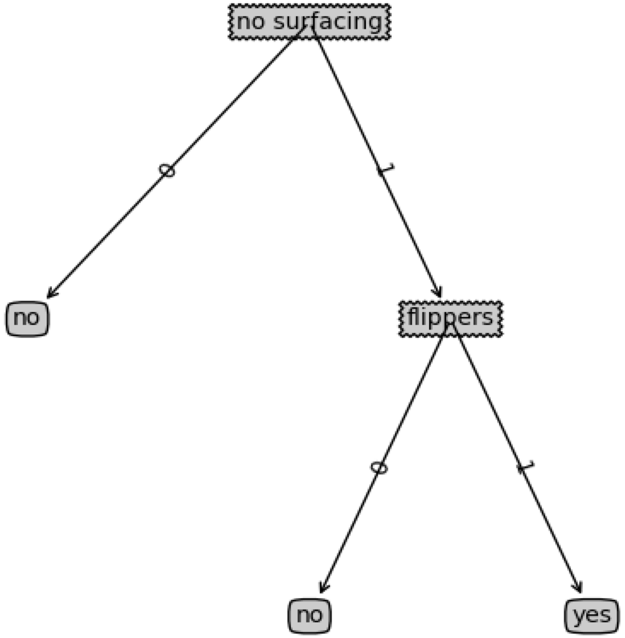
yOff = yOff - 1.0/totalD → 总层数

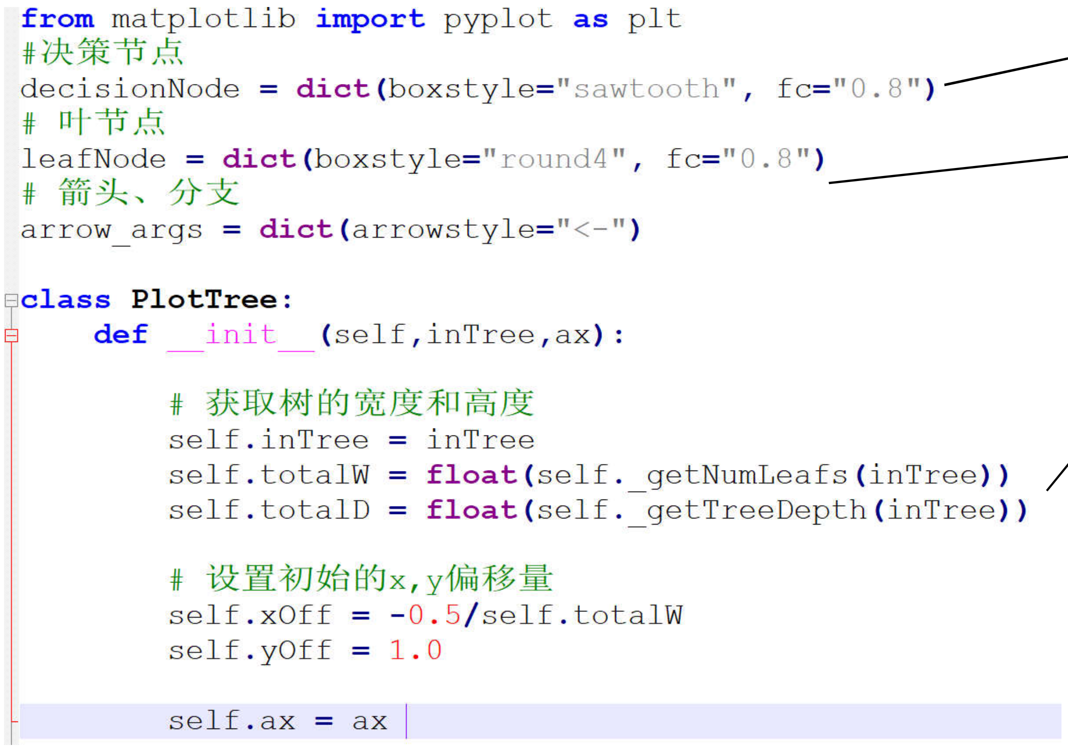
每画一个 叶子节点

xOff = xOff + 1.0/totalW → 总叶子数

decisionNode的位置

xOff +1/totalW /2 + (numLeafs/totalW)/2



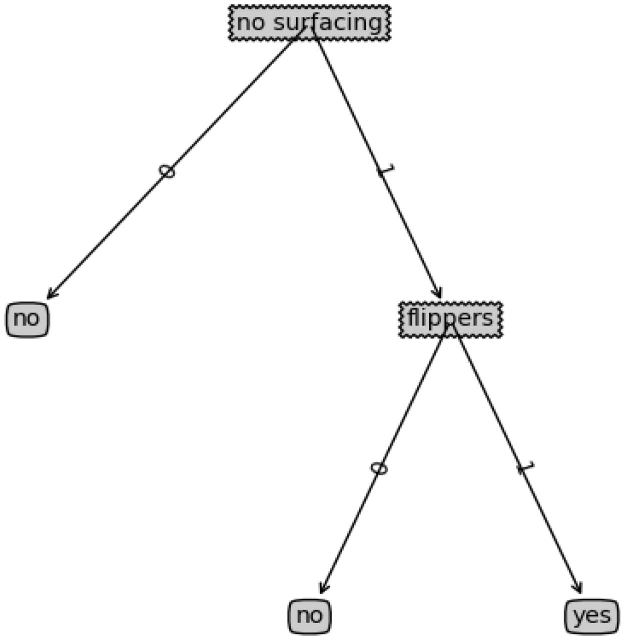


```
from matplotlib import pyplot as plt
#决策节点
decisionNode = dict(boxstyle="sawtooth", fc="0.8")
# 叶节点
leafNode = dict(boxstyle="round4", fc="0.8")
# 箭头、分支
arrow_args = dict(arrowstyle="<-")

class PlotTree:
    def __init__(self,inTree,ax):

        # 获取树的宽度和高度
        self.inTree = inTree
        self.totalW = float(self._getNumLeafs(inTree))
        self.totalD = float(self._getTreeDepth(inTree))

        # 设置初始的x,y偏移量
        self.xOff = -0.5/self.totalW
        self.yOff = 1.0

        self.ax = ax
```

设置决策节点的样式

设置叶子结点的样式

计算整体的宽度和高度



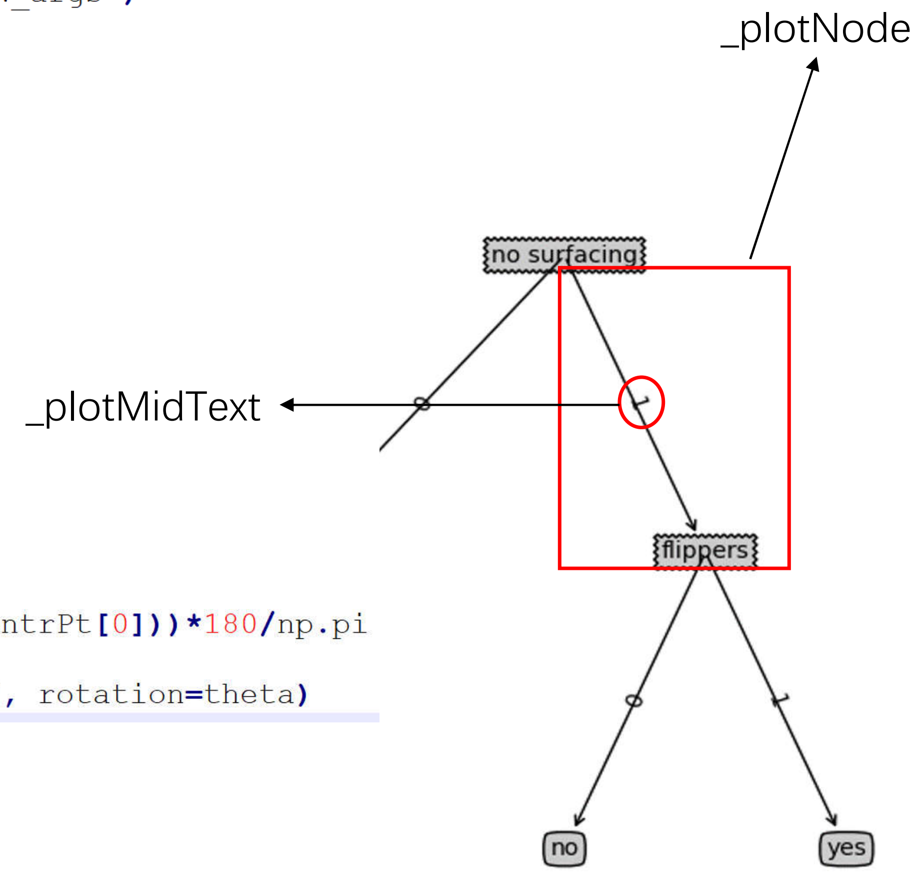
```
def _getNumLeafs(self,myTree):
    numLeafs = 0
    keys = myTree.keys()
    firstStr = list(keys)[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            numLeafs += self._getNumLeafs(secondDict[key])
        else:   numLeafs +=1
    return numLeafs
```

通过递归调用
计算树的宽度和高度

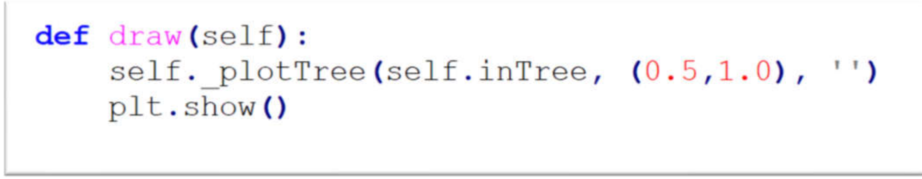
```
def _getTreeDepth(self,myTree):
    maxDepth = 0
    keys = list(myTree.keys())
    firstStr = keys[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__=='dict':
            thisDepth = 1 + self._getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth:
            maxDepth = thisDepth
    return maxDepth
```

```
def _plotNode(self,nodeTxt, centerPt, parentPt, nodeType):
    self.ax.annotate(nodeTxt, xy=parentPt,  xycoords='axes fraction',
         xytext=centerPt, textcoords='axes fraction',
         va="center", ha="center", bbox=nodeType, arrowprops=arrow_args )
```

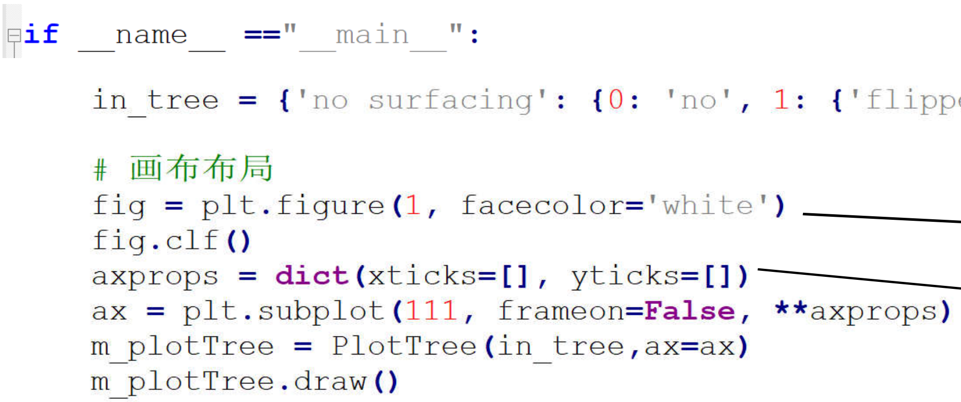
```
def _plotMidText(self,cntrPt, parentPt, txtString):
    # 获取 cntrPt, parentPt 的中点
    xMid =  (parentPt[0]+cntrPt[0])/2.0
    yMid = (parentPt[1]+cntrPt[1])/2.0

    # 计算 cntrPt、 parentPt 连线与水平方向的夹角
    if parentPt[0]-cntrPt[0] ==0:
        theta =90
    else:
        theta = np.arctan((parentPt[1]-cntrPt[1])/(parentPt[0]-cntrPt[0]))*180/np.pi

    self.ax.text(xMid, yMid, txtString, va="center", ha="center", rotation=theta)
```



```
def _plotTree(self,myTree, parentPt, nodeTxt):
    # 获取当前树的所有叶子节点的数目，即当前树的宽度
    numLeafs = self._getNumLeafs(myTree)

    keys = list(myTree.keys())
    firstStr = keys[0]

    # 当前节点的位置应该在所有当前树的中间
    cntrPt = (self.xOff + (1.0 + float(numLeafs))/2.0/self.totalW, self.yOff)
    self._plotMidText(cntrPt, parentPt, nodeTxt)
    plt.pause(1)
    self._plotNode(firstStr, cntrPt, parentPt, decisionNode)
    plt.pause(1)
    secondDict = myTree[firstStr]
    # 每画深一层 yOff减少
    self.yOff = self.yOff - 1.0/self.totalD
    for key in secondDict.keys():
        # 下一层是字典 画树
        if type(secondDict[key]).__name__=='dict':#test to see if the nodes are dictor
            self._plotTree(secondDict[key],cntrPt,str(key))        #recursion
        # 下一层是叶子
        else:
            # 每画一个叶子  xOff 增加
            self.xOff = self.xOff + 1.0/self.totalW
            self._plotNode(secondDict[key], (self.xOff, self.yOff), cntrPt, leafNode)
            plt.pause(1)
            self._plotMidText((self.xOff, self.yOff), cntrPt, str(key))
            plt.pause(1)
    # 返回一层 yOff增加
    self.yOff = self.yOff + 1.0/self.totalD
```



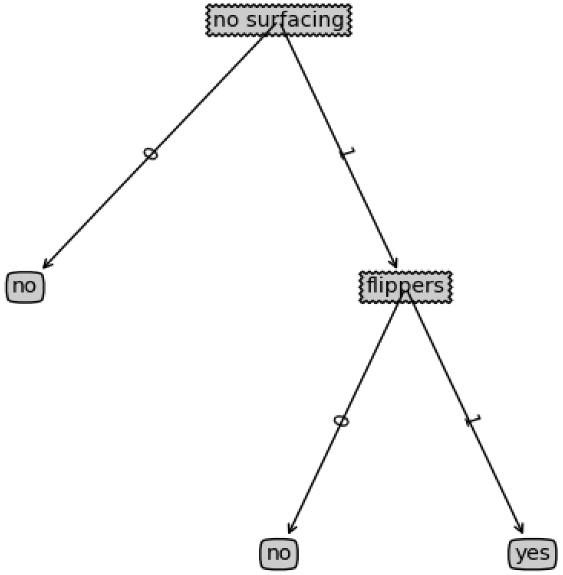
```
def draw(self):
    self._plotTree(self.inTree, (0.5,1.0), '')
    plt.show()
```



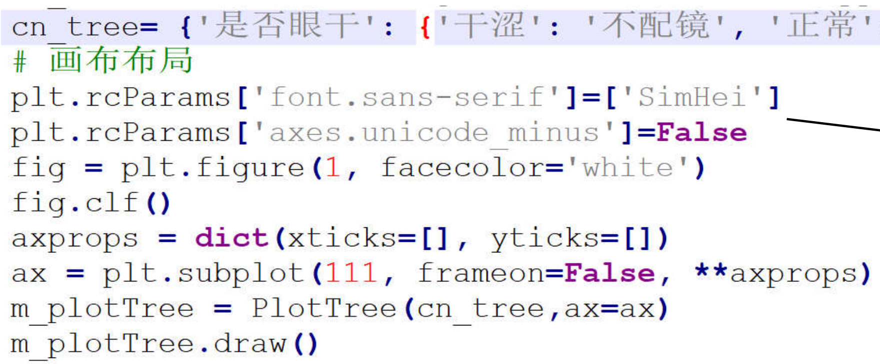
```
if __name__ =="__main__":

    in_tree = {'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}

    # 画布布局
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    ax = plt.subplot(111, frameon=False, **axprops)
    m_plotTree = PlotTree(in_tree,ax=ax)
    m_plotTree.draw()
```

画面布局，白色背景

没有横纵坐标



带有中文的情况:



```
cn_tree= {'是否眼干': {'干涩': '不配镜', '正常': {'是否散光':
# 画布布局
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False
fig = plt.figure(1, facecolor='white')
fig.clf()
axprops = dict(xticks=[], yticks=[])
ax = plt.subplot(111, frameon=False, **axprops)
m_plotTree = PlotTree(cn_tree,ax=ax)
m_plotTree.draw()
```

设置中文字体显示

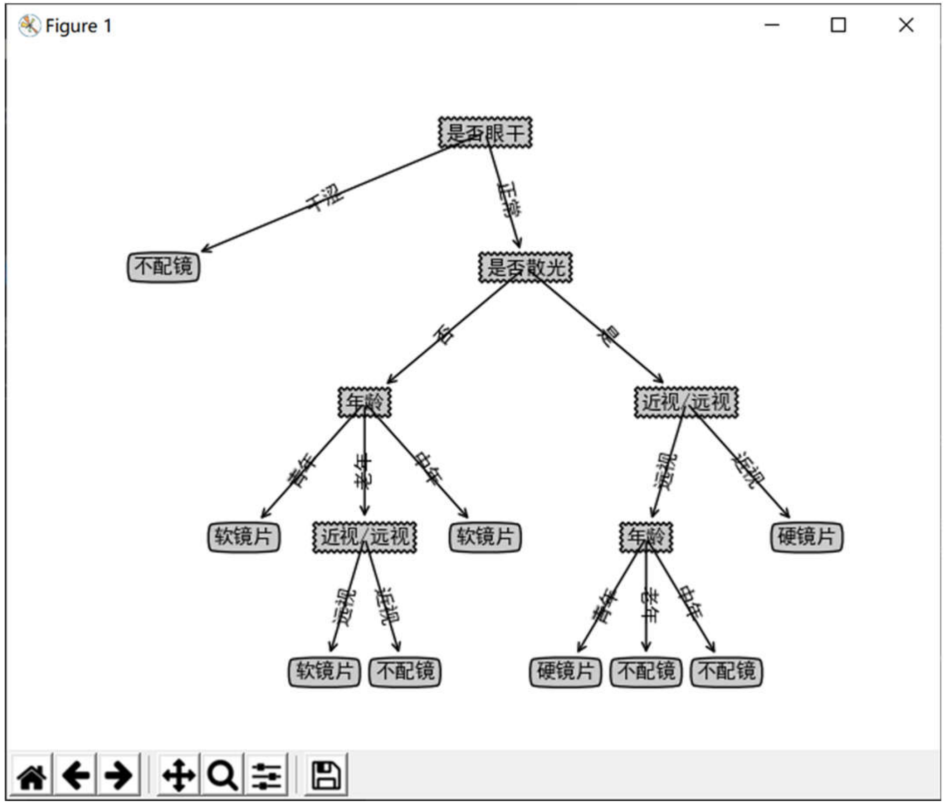


Figure 1
是否眼干
干涩
正常
不配镜
是否散光
否
是
年龄
近视/远视
青年
老年
中年
远视
近视
软镜片
近视/远视
软镜片
年龄
硬镜片
远视
近视
青年
老年
中年
软镜片
不配镜
硬镜片
不配镜
不配镜