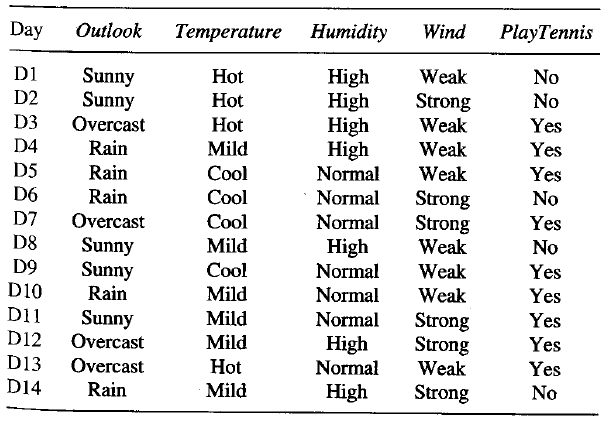
**实验1、监督学习**

**编程语言：**python3.6

**监督学习方法：**决策树

**所解决的实际问题：**根据湿度、天气、温度和风强决定是否出去打网球（老师讲课的实例）

**训练模型**



**基本思想：**利用python的字典结构形成一颗决策树，通过计算信息熵的增益确定根节点和非叶子节点

**具体过程和函数分析：**（语句含义有相应注释）

1. **创建决策树createTree(dataSet, labels)**

*#dataset是模型，labels是特征***def** createTree(dataSet, labels):*#创建树的函数* resultList = [example[-1] **for** example **in** dataSet]*#得到dataset最后一列的数据* **if** resultList.count(resultList[0]) == len(resultList): *# 该节点下所有结果相同停止划分* **return** resultList[0]  
 bestFeat = getBestFeature(dataSet)*#获得熵增加最多的特征* bestFeatLabel = labels[bestFeat]*#特征名称* myTree = {bestFeatLabel: {}}*#使用字典方式创建树* **del** (labels[bestFeat])*#删除被选为节点的特征* featValues = [example[bestFeat] **for** example **in** dataSet]*#得到选为节点的属性的属性值* uniqueVals = set(featValues)  
 **for** value **in** uniqueVals:*#通过递归得到每个属性值下的树* subLabels = labels[:] *# 为了不改变原始列表的内容复制了一下* myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet,bestFeat, value), subLabels)*#splitdataset是得到这个属性值下的所有模型的函数* **return** myTree

1. **获得信息增益最大的熵getBestFeature(dataSet)**

**def** getBestFeature(dataSet):  
 numFeats = len(dataSet[0]) - 1 *# 获得数据模型中属性的个数* baseEntropy = getShannonEnt(dataSet)*#计算基本的熵* bestInfoGain = 0.0  
 bestFeature = -1  
 **for** i **in** range(numFeats):*#对于模型中的每个属性* featList = [example[i] **for** example **in** dataSet]*#得到属性的所有属性值* uniqueVals = set(featList)  
 newEntropy = 0.0  
 **for** value **in** uniqueVals:*#对于每个属性值计算出他的熵* subDataSet = splitDataSet(dataSet, i, value)*#得到这个属性值对应的模型* prob = len(subDataSet) / float(len(dataSet))  
 newEntropy += prob \* getShannonEnt(subDataSet)*#相加得到这个属性的熵* infoGain = baseEntropy - newEntropy*#两个熵相减得到信息增益* **if** infoGain > bestInfoGain:*#比较信息增益得到信息增益最大的属性* bestInfoGain = infoGain  
 bestFeature = i  
 **return** bestFeature

1. **计算熵的函数getShannonEnt(dataSet)**

*# 计算熵的函数***def** getShannonEnt(dataSet):  
 numEntries = len(dataSet)*#得到模型的大小* labelCounts = {}*#结果的词典* **for** feaVec **in** dataSet:*#对于模型中的每条数据* currentLabel = feaVec[-1]  
 **if** currentLabel **not in** labelCounts:  
 labelCounts[currentLabel] = 0*#如果是一个新的结果，将结果加入到字典的key中* labelCounts[currentLabel] += 1*#对应key的值+1* shannonEnt = 0.0  
 **for** key **in** labelCounts:*#计算熵* prob = float(labelCounts[key]) / numEntries  
 shannonEnt -= prob \* log(prob, 2)  
 **return** shannonEnt

1. **挑出某个属性值下的模型def splitDataSet(dataSet, i, value)**

**def** splitDataSet(dataSet, i, value):*#dataset是原模型，i是属性的标号，value是具体的属性值* DataSet = []  
 **for** feat **in** dataSet:*#对于每条模型，如果属性值相同，就加入到这个属性列表中去* **if** feat[i] == value:  
 DataSet.append(feat[i + 1:])  
 **return** DataSet

1. **得到决策结果的函数getresult(pathtree)**
2. **主函数**

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 data = [[1, 1, 1, 1, **'no'**],*#初始化训练模型* [1, 1, 1, 2, **'no'**],  
 [2, 1, 1, 1, **'yes'**],  
 [3, 2, 1, 1, **'yes'**],  
 [3, 3, 2, 1, **'yes'**],  
 [3, 3, 2, 2, **'no'**],  
 [2, 3, 2, 2, **'yes'**],  
 [1, 2, 1, 1, **'no'**],  
 [1, 3, 2, 1, **'yes'**],  
 [3, 2, 2, 1, **'yes'**],  
 [1, 2, 2, 2, **'yes'**],  
 [2, 2, 1, 2, **'yes'**],  
 [2, 1, 2, 1, **'yes'**],  
 [3, 2, 1, 2, **'yes'**]]  
 labels = [**'OUTLOOK'**, **'TEMPERATURE'**, **'HUNIDITY'**, **'WIND'**, **'PLAYTENNIS'**]*#属性标签* myTree = createTree(data, labels)*#得到树* print(myTree)*#输出树的结构* print(**'OUTLOOK:SUNNY-1 OVERCAST-2 RAIN-3'**)  
 print(**'TEMPERATURE:HOT-1 MILD-2 COLL-3'**)  
 print(**'HUMIDITY:HIGH-1 NORMAL-2'**)  
 print(**'WIND:WEAK-1 STRONG-2'**)  
 pathtree=myTree  
 result=getresult(pathtree)*#得到结果* print(result)*#输出结果*

**def** getresult(pathtree):*#得到最终结果的函数* **for** key **in** pathtree.keys():*#从根节点开始询问* print(**'how about the'**,key)  
 value=input()  
 value=int(value)*#得到选择* **if** pathtree[key][value]==**'yes'**:*#如果字典对应的值是'yes'或者'no'到达叶节点* **return 'yes'  
 elif** pathtree[key][value]==**'no'**:  
 **return 'no'  
 return** getresult(pathtree[key][value])*#否则的话看这个节点下面的树，递归得到应得的结果*

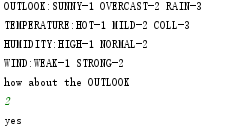
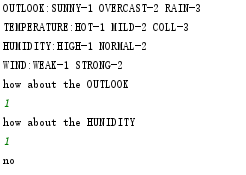
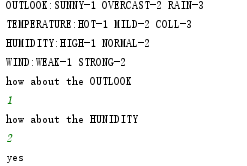
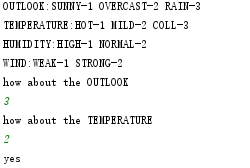
**实验结果：**

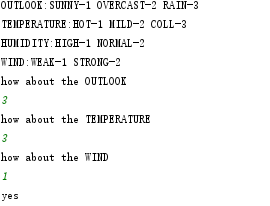
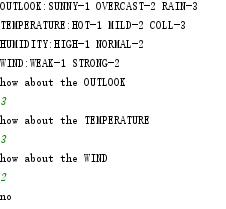
**该训练模型下得到的树：**

{'OUTLOOK': {1: {'HUNIDITY': {1: 'no', 2: 'yes'}}, 2: 'yes', 3: {'TEMPERATURE': {2: 'yes', 3: {'WIND': {1: 'yes', 2: 'no'}}}}}}

可以看出OUTLOOK是根节点，三个分支分别为HUNIDITY非叶节点、叶节点‘yes’和非叶节点TEMOERATURE，TEMOERATURE节点下有叶节点’yes’和非叶节点WIND。

对于不同的输入得到不同的结果，以下为结果截图：

**实验感想：**

决策树在实际中有很广泛的应用，这次实验使我加深了对决策树的理解，最大的收获其实是对于python语言中字典的使用，尤其是决策树在python语言中的表示形式。

**实验2、非监督学习**

**编程语言：**python

**监督学习方法：**k-means

**所解决的实际问题：**将30张带有人脸的图片分成三类

**训练模型：**30张带有人脸的图片，1-10为刘诗诗，11-20为奥巴马，21-30为洪金宝。



**基本思想：**利用python的dlib库获得每张图片人脸特征点坐标形成一个多维向量，这个库在的形成向量的函数只能在python3.5以上使用，将图片名称和对应的向量存储在字典中，利用K-means方法进行聚簇，以向量间的欧氏距离为距离。

**具体过程和函数分析：**（语句含义有相应注释）

1. **获得人脸向量的函数getFace()**

**def** getFace():  
 *# 加载人脸检测器，人脸模型，人脸特征点构建模型* detector = dlib.get\_frontal\_face\_detector()  
 facerec = dlib.face\_recognition\_model\_v1(**"2.dat"**)  
 predictor = dlib.shape\_predictor(**"1.dat"**)  
 i=1  
 dic={}  
 **for** f **in** glob.glob(os.path.join(**'./face'**, **"\*.jpg"**)):  
 image = cv2.imread(f)*#读取图片* dets = detector(image, 1)*#人脸检测* **for** k, d **in** enumerate(dets):  
 *# 2.关键点检测* shape = predictor(image, d)  
 *# 3.描述子提取，128D向量* u\_1 = facerec.compute\_face\_descriptor(image, shape)*#形成向量* u=numpy.array(u\_1)  
 dic[i]=u*#将向量加入到字典中* i+=1  
 **return** dic

1. **进行k-means迭代的函数**kmeans(s,dic)

**def** kmeans(s,dic):*#s存放原始簇，dic是所有数据的字典* s1=[[],[],[]]*#初始化二维列表* s.sort()*#将原来的簇排序* t=0  
 mid={}*#存放簇均值的字典* **for** i **in** s:*#计算簇均值* t+=1  
 m=0  
 cc = numpy.zeros(128)  
 **for** c **in** i:  
 cc+=dic[c]  
 m+=1  
 cc=[x/m **for** x **in** cc]  
 mid[t]=cc*#将簇的均值存放到字典中* t=0  
 distance=numpy.zeros(4)*#存放距离的数组* **for** key1 **in** dic.keys():*#对于所有的数据* t+=1  
 **for** i **in** mid.keys():*#计算与三个簇均值的距离* distance[i]=numpy.linalg.norm(dic[key1] - mid[i])*#欧几里得距离* h=1  
 mind=1000  
 minh=0  
 **while**(h<4):*#选取距离最小的簇* **if** distance[h]<mind:  
 minh=h-1  
 mind=distance[h]  
 h+=1  
 s1[minh].append(key1)*#加入到距离最小的簇中* s1.sort()*#将形成的新的三个簇排序* **if** s==s1:*#如果和原来的簇相等停止递归* **return** s1  
 **else**:*#否则再次迭代* **return** kmeans(s1,dic)

1. **主函数**

**实验结果：**

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 dic=getFace()*#获得人脸字典* s=[[1],[2],[3]]*#选取初始簇* s=kmeans(s,dic)*#k-均值聚类* print(s)*#输出*

当选取第1、2、3张图片为初始簇时得到的结果如下：

[[1, 2, 12, 19, 23, 25, 26, 27, 28, 29, 30], [3, 4, 5, 6, 7, 8, 9, 10, 11, 13], [14, 15, 16, 17, 18, 20, 21, 22, 24]]

可以看出第一个簇中有7张洪金宝的图片，第二个簇中有8张刘诗诗的图片，第三个簇中有6张奥巴马的图片。聚簇的效果不是特别的理想。

当选取第11、12、13张图片为初始簇时得到的结果如下：

[[1, 2, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30], [3, 4, 5, 6, 7, 8, 9, 10, 11], [13]]

可以看出10张洪金宝的图片都在第一个簇中，这个簇中另外10张图片中有8张奥巴马，两张刘诗诗，第二个簇中有8张刘诗诗，一张奥巴马，最后一个簇中只有一张奥巴马。

以上两种情况说明初始簇的选取会影响到最终聚簇的结果。

我尝试找出聚簇不理想的原因，对程序进行调试，调试过程中发现我的k-means（）函数并没有错误，造成问题的原因在于向量的选取，我使用的dlib库和人脸模型对于人脸识别后形成的多维向量是存在误差的，就算对于同一张图片进行识别得到的向量也会有不同，如果要优化可以从这方面进行优化，但目前本人水平和时间有限所以暂不做优化。

**实验感想：**

Kmeas聚簇方法对于距离的计算方法和向量的确定有很大的要求，在使用这个方法时向量的确定应当尽可能准确，同时要选择适当的距离计算方法，另外初始簇的选择也对结果又很大的影响，应当多次实验确定最优的初始簇。