桂林电子科技大学2021-2022学年 第2学期

**机器学习 实验报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验名称 | **实验一、线性回归与决策树** | | |  |
| 学 号 | **1901510216** | 姓名 | **李赏** |

**一、实验目的**

（1）掌握线性回归算法和决策树算法ID3的原理

（2）学会线性回归算法和决策树算法 ID3 的实现和使用方法。

**二、实验内容**

**1.线性回归模型实验**

（1）假设 line-ext.csv 是对变量 y 随变量 x 的变化情况的统计数据集。请根据教材的公式（3.7，3.8），使用 Python 语言编程计算线性回归模型的系数，建立一个线性回归模型。要求：1）计算出回归系数，输出模型表达式，绘制散点图和回归直线，输出均方误差。2）给出当自变量 x =0.8452 时，因变量 y 的预测值。

（2）对于上面的数据集 line-ext.csv，分别使用第三方模块 sklearn 模块中的类LinearRegression、第三方模块 statsmodels 中的函数 ols()来计算线性回归模型的系数，建立线性回归模型，验证上面计算结果及预测结果。

**2. 决策树算法实验**

（1）隐形眼镜数据集 glass-lenses.txt 是著名的数据集。它包含了很多患者眼部状况的观察条件以及医生推荐的隐形眼镜类型。请使用 Python 语言建立决策树模型ID3，划分25%的数据集作为测试数据集，并尝试进行后剪枝操作。要求1）使用 Graphviz 工具，将此决策树绘制出来。2）输出混淆矩阵和准确率。

（2）对于数据集glass-lenses.txt，使用sklearn模块中的类DecisionTreeClassifier（其中，criterion=’entropy’）来建立决策树模型 ID3，重复 2（1）中的操作，验证上面计算结果。

**三、实验代码和过程**

**1.线性回归模型实验**

**（1）**

1、读入数据集

|  |
| --- |
| # 从CSV文件中读取数据，并返回2个数组。分别是自变量x和因变量y。方便TF计算模型。  def zc\_read\_csv():  zc\_dataframe = pd.read\_csv("line-ext.csv", sep=",")  x = []  y = []  for zc\_index in zc\_dataframe.index:  zc\_row = zc\_dataframe.loc[zc\_index]  x.append(zc\_row["YearsExperience"])  y.append(zc\_row["Salary"])  return (x,y)  x, y = zc\_read\_csv() |

2、计算回归系数，并输出模型表达式

|  |
| --- |
| # 计算回归系数  averageX = 0  averageY = 0  for i in range (len(x)):  averageX += x[i]  for i in range (len(y)):  averageY += y[i]  #先计算平均数  averageX = averageX/(len(x))  averageY = averageY/(len(y))  #计算回归系数W  numerator = 0  denominator = 0  twice = 0  for i in range (len(x)):  numerator+=(y[i]\*(x[i]-averageX))  twice += (x[i]\*x[i])  denominator +=x[i]  decominator = math.pow(denominator,2)/(len(x))  w = numerator/(twice-decominator)  b=0  for i in range (len(x)):  b +=(y[i]-w\*x[i])  b /= (len(x))  print("系数为",w)  print("偏置为",b)  s="模型表达式y="+repr(w)+"x+"+repr(b)  print(s) |

3.绘制图像

|  |
| --- |
| # 获得画图对象。  fig = plt.figure()  fig.set\_size\_inches(20, 8) # 整个绘图区域的宽度10和高度4  ax = fig.add\_subplot(1, 2, 1) # 整个绘图区分成一行两列，当前图是第一个。  # 画出原始数据的散点图。  ax.set\_title("salary table")  ax.set\_xlabel("YearsExperience")  ax.set\_ylabel("Salary")  ax.scatter(x, y)  plt.plot([0,1],[3.0077432426975914,4.7030658848688107],c='green',ls='-')  plt.show() |

4.计算方差

|  |
| --- |
| # 计算方差  mse = 0  for i in range(len(x)):  mse += ((x[i]\*w+b)-y[i])\*((x[i]\*w+b)-y[i])  mse /= (len(x))  print("线性回归方程的均方误差：",mse) |

**（2）**

1. LinearRegression预测

|  |
| --- |
| # create model instance  linearModel = LinearRegression()  # train fit  linearModel.fit(x\_train, y\_label)  print("use sklearn w= {} b= {}".format(linearModel.coef\_, linearModel.intercept\_))  print("predict x=0.8452 y={} ", linearModel.predict(  np.array([0.845]).reshape(-1, 1))) |

2. statsmodels函数OLS()计算

|  |
| --- |
| X = sm.add\_constant(x\_train)  statsmodels\_model = sm.OLS(y\_label, X)  results = statsmodels\_model.fit()  print("use statsmodels w= {} b= {}".format(  results.params[1], results.params[0]))  print("result：y={}x+{}".format(  results.params[1], results.params[0])) |

**2.决策树**

①数据准备

|  |
| --- |
| dic={'young':0, 'pre':1, 'presbyopic':2, 'myope':0, 'hyper':1, 'yes':1, 'no':-1, 'reduced':-1, 'normal':1, 'no lenses':0, 'soft':1, 'hard':2}  #转换函数  def convert(v):  return dic[v]  def load\_data(file):  data = pd.read\_csv(file,sep='\t', header=0, names=['age', 'spectacle', 'astigmatic', 'tear','class'],index\_col=False)  data = data.applymap(convert)  return data  data = load\_data('glass-lenses.txt')  print(data) |

②计算信息熵、条件信息熵、信息增益

|  |
| --- |
| # 数据集信息熵H(D)  def entropyOfData(D):  nums = len(D)  cate\_data = D['class'].value\_counts()  cate\_data = cate\_data / nums # 类别及其概率  # 计算数据集的熵  return sum(- cate\_data \* np.log2(cate\_data))  # 条件信息熵H(D|A)  def conditional\_entropy(D, A):  # 特征A的不同取值的概率  nums = len(D) # 数据总数  A\_data = D[A].value\_counts()# 特征A的取值及个数  A\_data = A\_data / nums # 特征A的取值及其对应的概率  # print(A\_data)  # 特征A=a的情况下，数据集的熵  D\_ik = D.groupby([A, 'class']).count().iloc[:, 0].reset\_index() # Dik的大小  D\_i = D.groupby(A).count().iloc[:, 0] # Di的大小  # print(D\_ik)  # print(D\_i)  # 给定特征A=a的情况下，数据集类别的概率  D\_ik['possibilities'] = D\_ik.apply(lambda x: x[2]/D\_i[x[0]], axis=1)  # print(D\_ik)  # 给定特征A=a的情况下，数据集的熵  entropy = D\_ik.groupby(A)['possibilities'].apply(lambda x: sum(- x \* np.log2(x)))  # print(entropy)  # 计算条件熵H(D|A)  return sum(A\_data \* entropy)  # print(conditional\_entropy(data, 'age'))  # 信息增益  def info\_gain(D, A):  return entropyOfData(D) - conditional\_entropy(D, A) |

③选择最优特征属性

|  |
| --- |
| # 获得最优特征  def getOptiFeature(D):  Max = -1  Opti = ''  for A in list(D)[:-1]:  cur = info\_gain(D, A)  if Max < cur:  Opti = A  Max = cur  return Opti, Max  # 选择最优特征  print(getOptiFeature(data)) |

④划分数据集，构建决策树

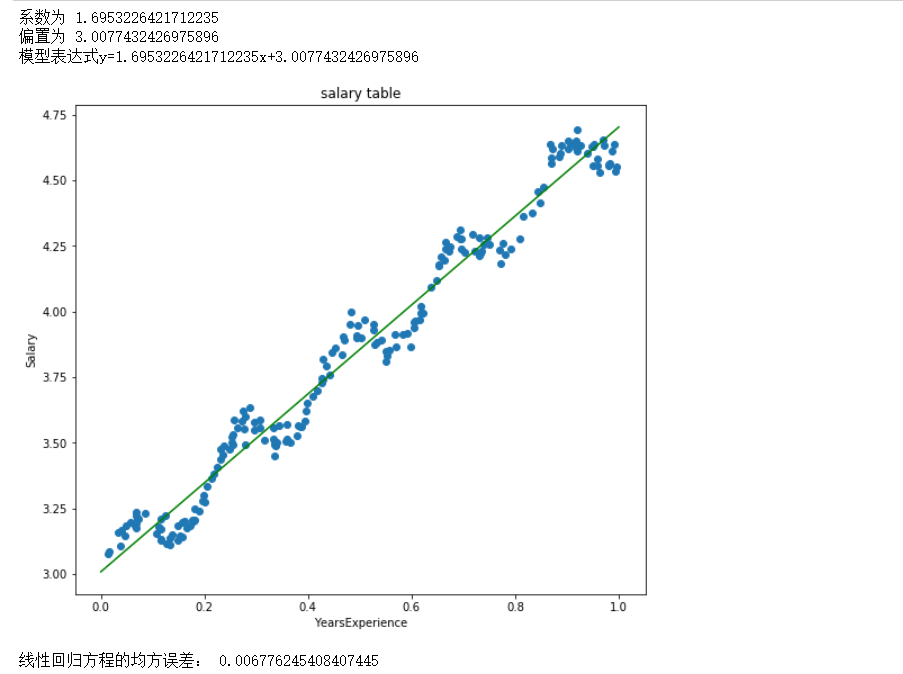
|  |
| --- |
| def majorityCnt(classList):  classCount = {}  for vote in classList: # 统计当前划分下每中情况的个数  if vote not in classCount.keys():  classCount[vote] = 0  classCount[vote] += 1  sortedClassCount = sorted(classCount.items, key=operator.itemgetter(  1), reversed=True) # reversed=True表示由大到小排序  # 对字典里的元素按照value值由大到小排序  return sortedClassCount[0][0]  def createTree(dataSet, labels):  classList = [example[-1]  for example in dataSet] # 创建数组存放所有标签值,取dataSet里最后一列（结果）  # 类别相同，停止划分  # 判断classList里是否全是一类，count() 方法用于统计某个元素在列表中出现的次数  if classList.count(classList[-1]) == len(classList):  return classList[-1] # 当全是一类时停止分割  # 长度为1，返回出现次数最多的类别  if len(classList[0]) == 1: # 当没有更多特征时停止分割，即分到最后一个特征也没有把数据完全分开，就返回多数的那个结果  return majorityCnt(classList)  # 按照信息增益最高选取分类特征属性  bestFeat = chooseBestFeatureToSplit(dataSet) # 返回分类的特征序号,按照最大熵原则进行分类  bestFeatLable = labels[bestFeat] # 该特征的label, #存储分类特征的标签  myTree = {bestFeatLable: {}} # 构建树的字典  del(labels[bestFeat]) # 从labels的list中删除该label  featValues = [example[bestFeat] for example in dataSet]  uniqueVals = set(featValues)  for value in uniqueVals:  # 子集合 ,将labels赋给sublabels，此时的labels已经删掉了用于分类的特征的标签  subLables = labels[:]  # 构建数据的子集合，并进行递归  myTree[bestFeatLable][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLables)  return myTree  def classify(inputTree, featLabels, testVec):  firstStr = next(iter(inputTree)) # 根节点  secondDict = inputTree[firstStr]  featIndex = featLabels.index(firstStr) # 根节点对应的属性  classLabel = None  for key in secondDict.keys(): # 对每个分支循环  if testVec[featIndex] == key: # 测试样本进入某个分支  if type(secondDict[key]).\_\_name\_\_ == 'dict': # 该分支不是叶子节点，递归  classLabel = classify(secondDict[key], featLabels, testVec)  else: # 如果是叶子， 返回结果  classLabel = secondDict[key]  return classLabel |

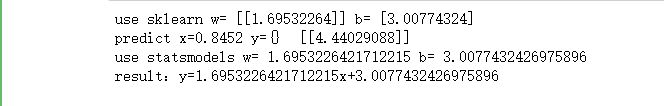
⑤绘制决策树

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  with open('glass-lenses.txt', 'r') as fr\_train: # 加载文件  lenses\_train = [inst.strip().split('\t') for inst in fr\_train.readlines()] # 处理文件  lenses\_target\_train = [] # 提取每组数据的类别，保存在列表里  for each in lenses\_train:  lenses\_target\_train.append(each[-1])  lensesLabels\_train = ['age', 'prescript', 'astigmatic', 'tearRate'] # 特征标签  lenses\_list\_train = [] # 保存lenses数据的临时列表  lenses\_dict\_train = {} # 保存lenses数据的字典，用于生成pandas  for each\_label in lensesLabels\_train: # 提取信息，生成字典  for each in lenses\_train:  lenses\_list\_train.append(each[lensesLabels\_train.index(each\_label)])  lenses\_dict\_train[each\_label] = lenses\_list\_train  lenses\_list\_train = []  #print(lenses\_dict) # 打印字典信息  lenses\_pd\_train = pd.DataFrame(lenses\_dict\_train) # 生成pandas.DataFrame  #print(lenses\_pd\_train)  le\_train = preprocessing.LabelEncoder() # 创建LabelEncoder()对象，用于序列化  for col in lenses\_pd\_train.columns: # 为每一列序列化  lenses\_pd\_train[col] = le\_train.fit\_transform(lenses\_pd\_train[col])  #print(lenses\_pd\_train)  clf = tree.DecisionTreeClassifier(criterion='entropy',max\_depth=4) # 创建DecisionTreeClassifier()类  clf = clf.fit(lenses\_pd\_train.values.tolist(), lenses\_target\_train) # 使用数据，构建决策树  print(clf)  dot\_data = StringIO()  tree.export\_graphviz(clf, out\_file=dot\_data, # 绘制决策树  feature\_names=lenses\_pd\_train.keys(),  class\_names=clf.classes\_,  filled=True, rounded=True,  special\_characters=True)  graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())  graph.write\_pdf("tree.pdf") |

## ****四、实验结果截图及结果分析****

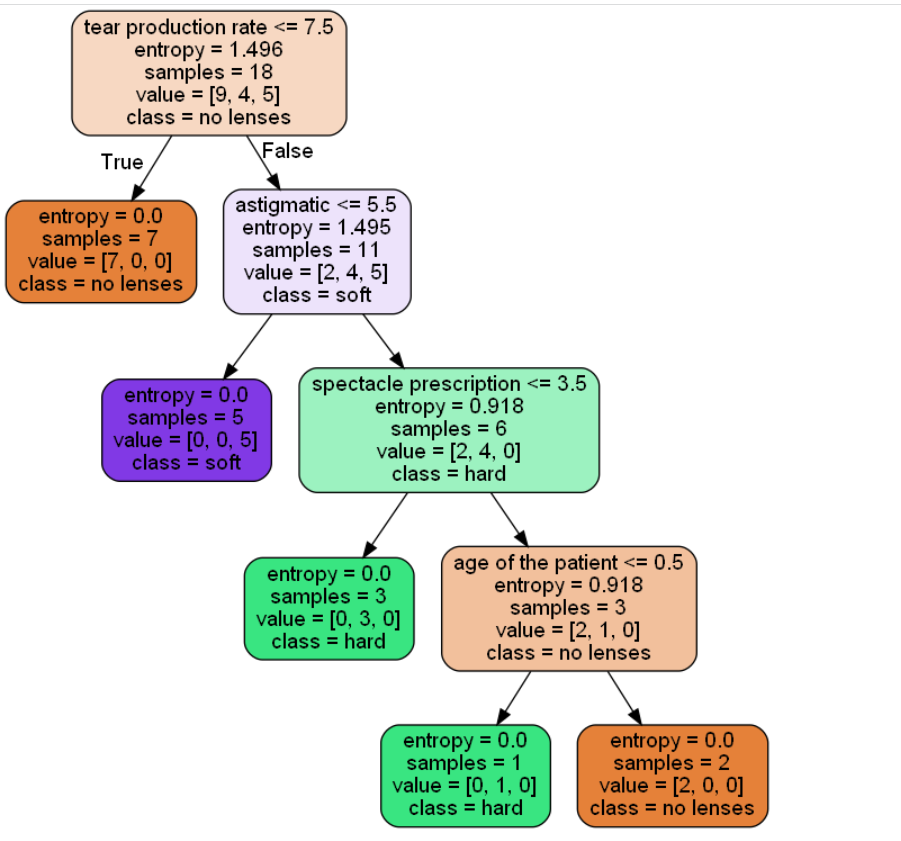
**1.线性回归模型实验**





**2.决策树算法实验**

决策树绘制



混淆矩阵和准确率



## ****五、实验中遇到的问题和解决问题的方法****

决策树部分的理论知识不太熟悉，同时代码实现上也遇到了许多问题，涉及决策树绘制及混淆矩阵输出部分没有接触过，相关库也是第一次使用，通过查询资料，大部分代码问题可以解决。今后决策树的理论知识还需要多加掌握。