数据分析第二次作业(孙万彤)

一．线性回归

1. 概念

线性回归是利用数理统计中回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法，运用十分广泛。其表达形式为 y = w'x+e，e 为误差服从均值为 0 的正态分布。线性回归假设特征和结果满足线性关系。其实线性关系的表达能力非常强大，每个特征对结果的影响强弱可以由前面的参数体现，而且每个特征变量可以首先映射到一个函数，然后再参与线性计算。这样就可以表达特征与结果之间的非线性关系。 用 X1，X2..Xn 去描述 feature 分量，我们可以做出一个估计函数：

$$h(x) = h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

如果我们令 X0 = 1，就可以表示为如下的向量形式：

$$h_\theta(x) = \theta^T X$$

程序也需要一个机制去评估我们 θ 是否比较好，需要对我们做出的 h 函数进行评估，这个函数就被称为损失函数（loss function）或者错误函数(error function)，用函数名为的 J 函数描述 h 函数偏离的程度

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\min_\theta J_\theta$$

这个错误估计函数是去对 x(i)的估计值与真实值 y(i)差的平方和作为错误估计函数，至于为何选择平方和作为错误估计函数，从概率分布的角度讲解了该公式的来源。如何调整 θ 以使得 J(θ)取得最小值有很多方法，其中有最小二乘法(min square)，和梯度下降法。

(1).最小二乘法

最小二乘法是基于均方误差最小化来进行模型求解的方法。最小二乘法就是试图找到一条直线，使的所有的样本到直线上的欧式距离之和最小。在用最小二乘法时必须要求 X 是满秩矩阵

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

(2).梯度下降法

在机器学习算法中，在最小化损失函数时，可以通过梯度下降法来一步步的迭代求解，得到最小化的损失函数，和模型参数值。反过来，如果我们需要求解损失函数的最大值，这时就需要用梯度上升法来迭代了。梯度下降法和梯度上升法是可以互相转化的。比如我们需要求解损失函数f(θ)的最小值，这时我们需要用梯度下降法来迭代求解。但是实际上，我们可以反过来求解损失函数 -f(θ)的最大值，这时梯度上升法就派上用场了。

2. 算法原理及其实现(用最小二乘法)

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

在用 w 求值的时候由于大部分都是矩阵操作，所以首先求矩阵的转置矩阵

求转置矩阵

```
#求转置矩阵
def trans(array):
    valueMatrix = []
    spec = []
    j = 0
    for i in range(len(array[0])):
        for j in range(len(array)):
            valueMatrix.append(array[j][i])
            if len(valueMatrix) == len(array):
                spec.append(valueMatrix)
                valueMatrix = []
            j += 1
    return spec
```

求矩阵的乘积

```python
#求矩阵和矩阵的乘积
def getMatrixMuliply(traina,trainb):
    transsb = trans(trainb)
    matrixvalue = []
    templist = []
    sum = 0
    for x in traina:
        for y in transsb:
            sum = x[0]*y[0] + x[1]*y[1]
            templist.append(sum)
            if len(templist) == len(trainb[0]):
                matrixvalue.append(templist)
                templist = []
    return matrixvalue
```

求行列式的值

```python
#求行列式的值,默认按第一列展开
def getDetValue(nums):
    result = isinstance(nums, int)
    if result: return nums
    # print(nums[0])
    matLen = len(nums)
    if matLen == 1:
        if len(nums[0]) == 1:
            return nums[0][0]
        else:
            return nums[0]
    sum = 0.0
    #程序递归出口
    if matLen == 2:
        return nums[0][0]*nums[1][1] - nums[0][1]*nums[1][0]
    if matLen > 2:

        for i in range(matLen):  #按第一列展开
            #记录当前这个元素的行号和列号
            colums = 0
            rows = i
            newMat = []
            temp = []
            for j in range(matLen):
                if j != rows:
                    #newMat.append(nums[j])
                    for m in range(len(nums[j])):
                        if m != colums:
                            temp.append(nums[j][m])
                            if len(temp) == matLen - 1:
                                newMat.append(temp)
                                temp = []
            # print(newMat)
            sum += nums[i][0]*((-1)**((i+1)+1))*getDetValue(newMat)
        return sum
```

求矩阵伴随矩阵

```python
# 求矩阵的伴随矩阵
def getAdjointMatrix(nums):
    matLen = len(nums)
    # 放置求完之后的伴随矩阵
    adjoinMarix = []
    # 记录当前这个元素的列号
    colums = 0
    tempAdjoin = []
    if matLen > 0:
        for k in range(matLen):
            for i in range(matLen):  #
                # 记录当前这个元素的行号
                rows = i
                temp = []
                newMat = []
                for j in range(matLen):
                    if j != rows:  # 把同一行去掉
                        # newMat.append(nums[j])
                        for m in range(len(nums[j])):
                            if m != colums:  # 把同一列去掉
                                temp.append(nums[j][m])
                                if len(temp) == matLen - 1:  # 因为求代数余子式，所以减1,如果求伴随，则不减1
                                    newMat.append(temp)
                                    temp = []
                # 再次递归求代数余子式的值
                yuZiShi = ((-1) ** ((i + 1) + (colums + 1))) * getDetValue(newMat)

                tempAdjoin.append(yuZiShi)
                if len(tempAdjoin) == matLen:
                    adjoinMarix.append(tempAdjoin)
                    tempAdjoin = []
            colums += 1
        return adjoinMarix  # [15, -7, -2, -7, 3, 1, -2, 1, 0]
```

求逆矩阵

```python
#求逆矩阵(利用行变换求逆矩阵 返回求得的逆矩阵)
def inverseMatrix(trainSet=[]):
    # print(trainSet)
    # trainSet = plusParam(trainSet)
    #求得行列式的数值
    hangValue = getDetValue(trainSet)
    #判断行列式是否可逆,如果行列式的值不为零则可逆
    if hangValue == 0.0:
        return
    #求伴随矩阵
    banSuiSet = getAdjointMatrix(trainSet)
    #计算逆矩阵
    banSuiLen = len(banSuiSet)
    for i in range(banSuiLen):
        for j in range(banSuiLen):
            temp = banSuiSet[i][j] / hangValue
            banSuiSet[i][j] = float(temp)
    print(banSuiSet)
    #banSuiSet已经除去行列式的数值了
    inverMatrix = banSuiSet

    #返回逆矩阵
    return inverMatrix
```

读取本地数据源进行预处理

```
#读取本地数据源,然后进行数据预处理
def getDataSource():
    f = open("D://数据处理数据源//LR.txt", 'r')
    lines = f.readlines()
    train_x = []
    train_y = []
    #x矩阵的原始值，不添加1
    trainx_init = []
    trainy_init = []
    train_matri_y = []
    for line in lines:
        line = line.strip().split(",")
        #print(line)
        listTemp = []
        listTemp.append(int(line[0]))
        # print(line[0])
        trainx_init.append(int(line[0]))
        listTemp.append(int(line[1]))
        train_x.append(listTemp)


        train_y.append(int(line[2]))
        trainy_init.append(int(line[2]))
        # print(train_y)
        train_matri_y.append(train_y)
        train_y = []
    return train_matri_y,train_x,trainx_init
```

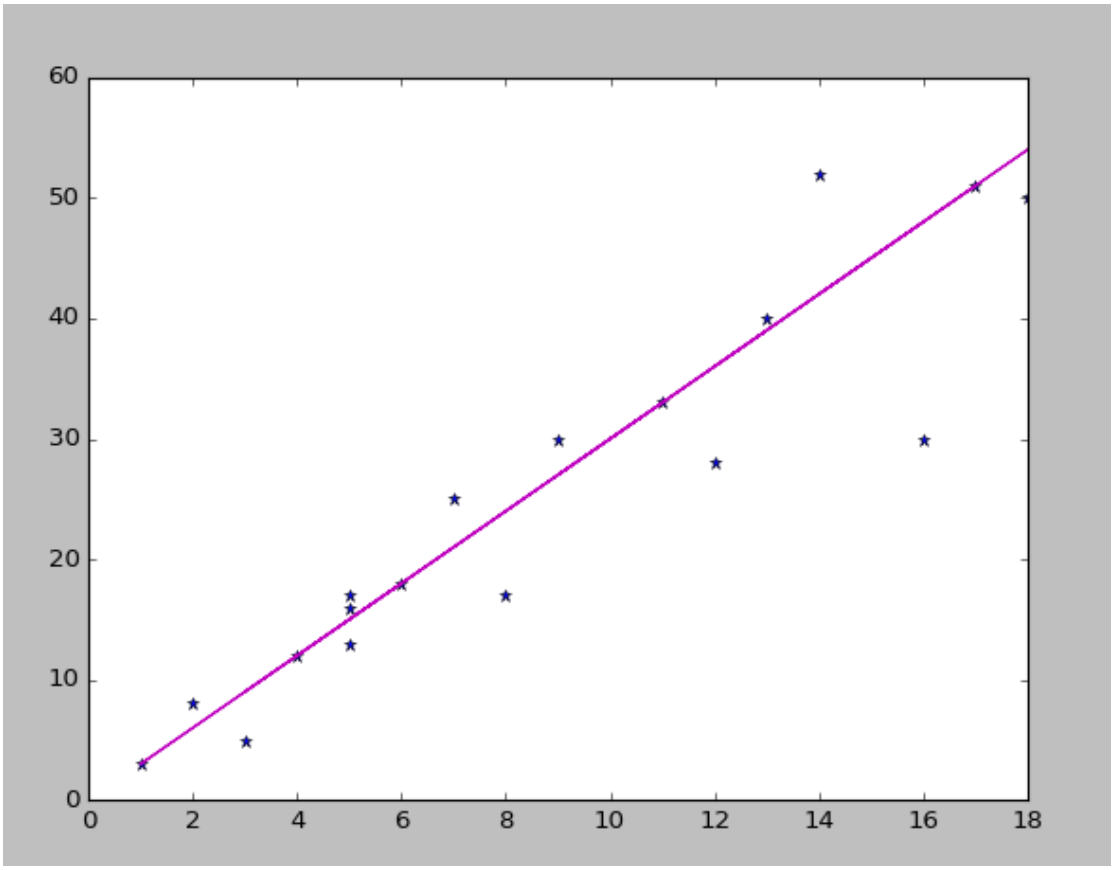
根据求得的样本特征绘制回归曲线的函数

```
def drawImages():
    #x和y的实际数值
    w = getLrParam()
    nomimal = getDataSource()
    real_x = nomimal[1]
    real_y = nomimal[0]

    real_x_init = nomimal[2]
    #y是m行1列
    predit_y = getMatrixMuliply(real_x,w)
    a = float(w[0][0])
    b = float(w[1][0])
    a = round(a,4)
    b = round(b,3)
    print("最终的回归直线为:y = %s*x + %s" % (a,b))

    # 画回归直线y = w0*x+w1
    #真实值
    plt.plot(real_x_init, real_y,"*")
    #预测数值
    plt.plot(real_x_init,predit_y,c='m')
    plt.show()
```

最终求得的线性回归的方程为:

```
最终的回归直线为:y = 3.0*x + 0.0
```

在二维平面上的表示为:



3. 总结

在用最小二乘法实现线性回归算法的时候，求矩阵的逆矩阵应该是最重要的一点，在矩阵的运算中，由于要经常进行类型的转换，所以造成均方误差可能表较大，这个还有待对代码进行优化。

二. 决策树算法实现及结果评估程序实现

1.概念

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直至到达叶子节点，将叶子节点存放的类别作为决策结果。 决策树的构造就是进行属性选择度量确定各个特征属性之间的拓扑结构。 构造决策树的关键步骤是分裂属性。所谓分裂属性就是在某个节点处按照某一特征属性的不同划分构造不同的分支，其目标是让各个分裂子集尽可能地“纯” 。尽可能“纯” 就是尽量让一个分裂子集中待分类项属于同一类别。当对属性进行分类时要判断属性属于连续值还是离散值，因为决策树只能对离散属性进行处理，所以就必须对数据进行预处理，对数据进行离散化或者装箱等的操作。

构造决策树的关键内容是进行属性选择度量，属性选择度量是一种选择分裂准则，是将给定的类标记的训练集合的数据划分 D “最好” 地分成个体类的启发式方法，它决定了拓扑结构及分裂点的选择

2.算法实现原理(c4.5)

a.选择属性的度量标准

(1) 信息熵

$$info(D) = -\sum_{i=1}^{m} p_i log_2(p_i)$$

其中 pi 表示第 i 个类别在整个训练数据集中出现的概率，用属于此类别元素的数量除以训练元组元素总数量作为估计。熵的实际意义表示是 D 中元组的类标号信息量的均值。

(2) 信息增益

现在我们假设将训练元组 D 按属性 A 进行划分，则 A 对 D 划分的期望信息为：

$$info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} info(D_j)$$

则信息增益为:

$$gain(A) = info(D) - info_A(D)$$

(3) 信息增益率

属性分裂时的信息为：

$$split\_info_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} log_2(\frac{|D_j|}{|D|})$$

增益率:

$$gain\_ratio(A) = \frac{gain(A)}{split\_info(A)}$$

3.算法源码

首先读取本地训练集，进行训练集的数据预处理

```
#读取训练集
def loadDataSet():
    f = open("D://数据处理数据源//j48_train.txt",'r')
    dataSet = f.readlines()
    lables = []
    dataMat = []
    attrValue = []
    for line in dataSet:
        line = line.strip().split(",")
        lables.append(line[-1])
        dataMat.append(line[0:(len(line))])
    attrValue = dataMat[0]
    dataMat.pop(0)
    lables.pop(0)
    attrValue.pop(-1)
    return lables,attrValue,dataMat
```

然后递归的建立决策树

```
#创建树
def createTree(dataSet,labels):
    classArray = [data[-1] for data in dataSet]
    if classArray.count(classArray[0]) == len(classArray):
        return classArray[0]

    if len(dataSet[0]) == 1:
        return majorityCount(classArray)

    #bestFeat返回字段在所在行的位置
    bestFeat = getBestSingleEntryDataSet(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    subLabels = labels[:]
    subLabels.pop(bestFeat)
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)

    for value in uniqueVals:
        #程序递归生成子数
        myTree[bestFeatLabel][value] = createTree(getSubDataSet(dataSet, bestFeat, value), subLabels)
    return myTree  #生成的树
```

其返回值为一个 dict 数据结构，里边存放的是分类树的一些信息，如下图所示：

然后读取本地测试集，递归的遍历树，对测试集合进行测试

```
#读取测试集
def loadTestDataSet():
    f = open("D://数据处理数据源//j48_test.txt",'r')
    dataSet = f.readlines()
    dataMat = []
    for line in dataSet:
        line = line.strip().split(",")
        dataMat.append(line)
    return dataMat
```

```python
#算法评价函数
def getEvaluateTarget(labels, myTree, testDataSet):
    TP,FP,FN,TN = 0,0,0,0
    presion,recall,f = 0,0,0
    for testData in testDataSet:
        #预测样本类别(返回预测后的样本的类别)
        des = classify(myTree,labels,testData)
        #实际值
        sign = testData[-1]
        # print(testData,des)
        if sign == '是':
            if des == '是':
                TP += 1
            else:
                FN += 1
        if sign == '否':
            if des == '是':
                FP += 1
            else:
                TN += 1
```

循环读取测试集，计算各项评价指标，需要遍历已经生成的树

```python
#遍历树
def classify(inputTree, featLabels, testVec):
    firstSides = list(inputTree.keys())
    firstStr = firstSides[0]
    classLabel = ''
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':
                classLabel = classify(secondDict[key],featLabels,testVec)
            else:
                classLabel = secondDict[key]
    return classLabel
```

在建立树的过程中计算各个属性熵值是比较重要的，需要递归的计算熵值，

```python
#计算单个属性的最大熵值，并返回，以便于进行数据集的划分
def getBestSingleEntryDataSet(dataSet):
    #获取特征个数
    attrNums = len(dataSet[0]) - 1  #去除最后一项标记类
    bestInfoGainRatio = 0.0
    baseEntropy = calEntropy(dataSet)
    # print(baseEntropy)
    bestFeature = -1
    #遍历所有的特征值
    for i in range(attrNums):
        listFeature = [data[i] for data in dataSet]
        #去除重复值
        uniqueVals = set(listFeature)

        tempEntropy = 0.0
        infoIV = 0.0
        for value in uniqueVals:
            #拿到特征值里边取某一个值的集合(为了计算熵方便)
            subDataSet = getSubDataSet(dataSet, i, value)
            probab = len(subDataSet) / float(len(dataSet))
            tempEntropy += probab * calEntropy(subDataSet)
            infoIV -= (probab * m.log(probab, 2))
        if infoIV != 0:
            infoGainRatio = (baseEntropy - tempEntropy) / infoIV
        else:
            infoGainRatio = 0
        if infoGainRatio > bestInfoGainRatio:
            bestInfoGainRatio = infoGainRatio
            #属性在数据集中列的位置
            bestFeature = i
    return bestFeature
```

最后得出的各项评价指标如下图所示:

```
Precision: 0.400000
Recall: 0.666667
F1: 0.500000
```

4.总结

在进行决策树的划分中过程中，本来是需要递归的计算熵值的，然而刚开始做的时候并没有递归导致决策树少了一些分支，从而导致在对测试集进行预测的时候有些预测集并没有预测出来，导致准确率 100%，后来改完之后则正常。另在测试的过程中，训练集和测试集划分是比较重要的，当用留余法的时候，如果使用单次划分，则模型的性能相对较差，需要多次随机划分数据集和训练集，最后综合每次的各种评价指标(取平均值)作为最后的评价指标。