**数据挖掘期末大作业情况说明**

数据挖掘期末考核形式为大作业，该环节总体分为两部分，一是现场报告，二是纸质作业。现场和作业各占总成绩50%。其中现场汇报设有汇报和评委老师提问环节，其中个人汇报占50%，回答问题占50%。最终期末大作业环节成绩总得分为：作业成绩（50%）+现场个人得分（50%）。

表1：个人表现打分明细表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 总体汇报讲述清晰明了（30%） | 知识理解准确及方法得当（30%） | 回答问题积极主动  （10%） | 回答内容完整并准确（30%） | 总分  （100分） |
|  |  |  |  |  |

评委（拟请）：郝建民、

附件：1.期末大作业文章格式



20\_\_— 20\_\_学年（\_）学期

大作业名称：基于线性回归和决策树对天气预报进行回归和分类

姓 名：\_\_\_inter\_\_\_\_\_

年 级: \_\_\_2021级\_\_\_\_\_

专业班级：\_\_\_21级大数据1班\_\_\_\_\_

学 号：\_\_\_202151018\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| **评分标准** | **各项标准满分** | **学生得分** | **总分** |
| 格式准确，语言通顺流畅，逻辑性强 | 10 |  |  |
| 设计目的明确 | 15 |  |
| 结果分析正确 | 15 |  |
| 对业务内容分析透彻合理、全面、详细 | 30 |  |
| 数据模型、算法模型运用得当，有实际价值 | 30 |  |
| **备注** | 如有大段抄袭或雷同文章均为不合格 | |

注:评分标准和各项标准满分仅供教师参考

目录

[一、总体介绍 1](#_Toc87903384)

[二、设计思路与方案 2](#_Toc87903385)

[三、模型说明 3](#_Toc87903386)

[四、实验运行结果及分析 4](#_Toc87903387)

[五、实验总结 5](#_Toc87903388)

[六、感想及体会 6](#_Toc87903389)

# 总体介绍

# 1.1项目介绍

天气预报是使用现代科学技术对未来某一地点地球大气层的状态进行预测。 从史前人类就已经开始对天气进行预测来相应地安排其工作与生活（比如农业生产、军事行动等等）。 今天的天气预报主要是使用收集大量的数据（气温、湿度、风向和风速、气压等等），然后使用目前对大气过程的认识（气象学）来确定未来空气变化。

该项目是一个经典的回归和分类问题，通过分析不同的因素来预测当前的温度。本次项目使用常用的回归模型线性回归和常用的分类模型决策树。使用合适的评价指标：准确率、精确率、召回率、F1-Score来评价模型的好坏。

**1.2 数据说明**

日期:当天的日期，唯一

天气状况: '多云/多云' '晴/晴' '雨夹雪/雨夹雪' '小雪/小雪' '多云/小雪' '多云/晴' '晴/多云' '小雪/多云' '多云/阴' '雨夹雪/多云' '阴/多云' '阴/小雨' '小雨/阴' '多云/小雨' '扬沙/晴' '扬沙/扬沙' '小雨/小雨' '阴/阴' '晴/小雨' '小雨/多云' '小雨/晴' '多云/雷阵雨' '晴/阴' '中雨/中雨' '雷阵雨/多云' '雷阵雨/雷阵雨' '小雨/大雨' '暴雨/暴雨' '暴雨/大雨'

气温:当天的气温范围。形如X/Y,X为最低温度，Y为最高温度

风力风向:形如东北风1-2级/北风1-2级

# 设计思路与方案

**2.1 问题描述**

在本项目中，需要解决以下问题：

0）数据爬取：使用requests，BeautifulSoup等常见库对天气后报网站进行页面分析，并根据xml结果写爬虫获取数据。

1）数据预处理：数据可能包括缺失值、异常值、重复值或者需要规范化的特征。需要对这些数据进行清洗和处理，以便更好地训练和预测。

2）特征工程：需要选择合适的特征，对特征进行转换、提取和选择，以更好地反映用户的行为和需求，提高模型的预测准确率。

3）模型调参：需要选择合适的分类算法和模型结构，并对模型进行调参，以达到最佳的性能和稳定性。

4）模型评价：需要使用合适的评价指标，对模型进行评估和比较，并对模型进行改进和优化。

**2.2 评价方法**

在本项目中，将使用以下评价方法：

1）准确率（Accuracy）：模型预测准确的用户数量除以总用户数。准确率是最简单且常用的评价方法。

2）精确率（Precision）：表示预测为流失用户的用户中实际为流失的比例。

3）召回率（Recall）：表示实际为流失用户的用户中被预测为流失的比例。

4）F1-Score：使用精确率和召回率的加权平均值，是两者综合评价的一个指标。

通过以上评价指标的选择和评估，可以优化模型的性能，提高客户流失预测的准确性。

# 三、模型说明

1.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。 回归分析中，只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。

2.决策树是一种基于树结构的机器学习模型，它用于分类和回归任务。在决策树中，每个内部节点表示一个特征或属性，每个分支代表一个决策规则，而每个叶节点表示一个输出标签或值。通过沿着树的分支从根节点到叶节点的路径，可以对输入数据进行分类或回归。

以下是决策树模型的一些关键概念和特点：

1. 节点（Node）：决策树的每个节点代表一个特征或属性，用于对输入数据进行分割。

2. 分支（Branch）：每个节点的分支表示一个决策规则，根据输入数据的特征值决定下一个节点。

3. 叶节点（Leaf Node）：叶节点是决策树的末端节点，表示模型对输入数据的最终输出。

4. 深度（Depth）：决策树的深度是从根节点到最深叶节点的路径长度。深度越大，模型越复杂。

5. 根节点（Root Node）：决策树的起始节点，它包含最重要的特征，用于开始对输入数据的分割。

6. 决策规则：决策树根据输入数据的特征值进行分割，每个分割都对应一个决策规则。

7. 信息增益（Information Gain）：用于在构建决策树时选择最佳分割特征的指标。信息增益越大，表示使用该特征进行分割后，对数据的纯度提升越大。

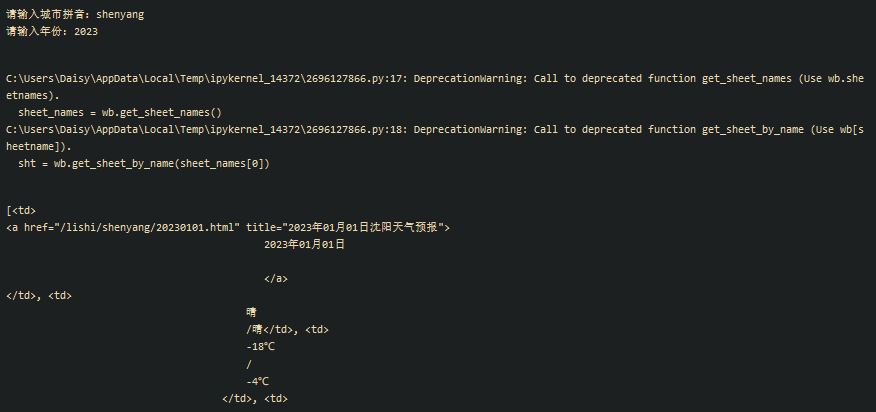
8. 过拟合（Overfitting）：当决策树过于复杂，以致于在训练数据上表现良好但在新数据上表现不佳时，就会发生过拟合问题。可以通过限制树的深度或剪枝来缓解过拟合。

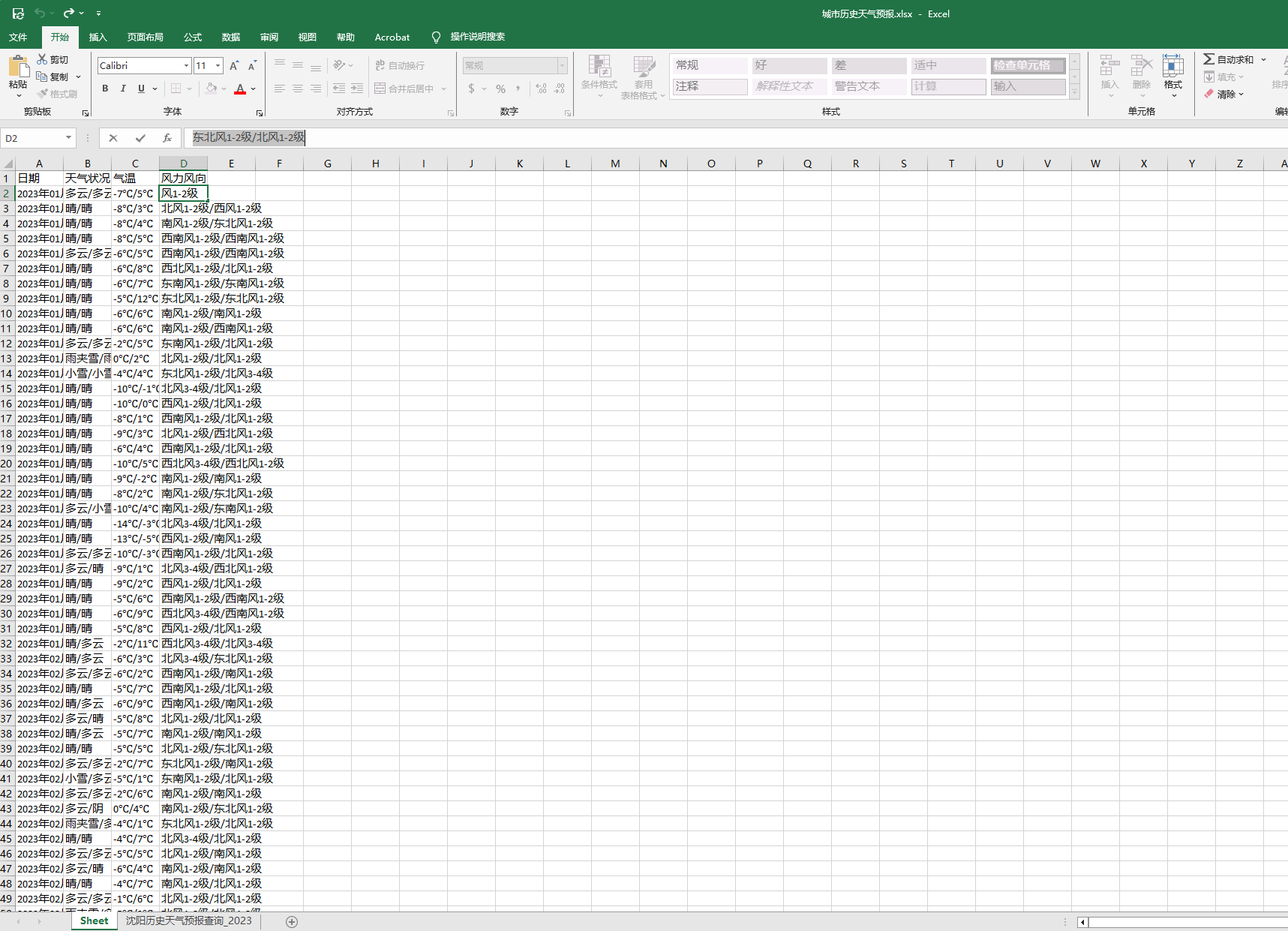
决策树的优点包括易于理解和解释，能够处理数值和类别型数据，对缺失值不敏感等。然而，决策树也容易受到数据噪声的影响，可能导致过拟合。

# 实验运行结果及分析

**1.实现爬虫**

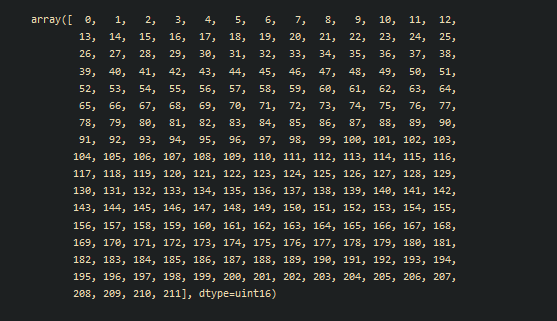
使用do while循环，输入城市名和年份，爬取当年全部天气数据，并使用正则表达式切分，直至输入‘n’该过程中止，原始数据保存在“城市历史天气预报.xlsx”

****

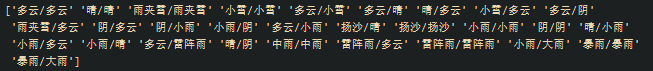
****

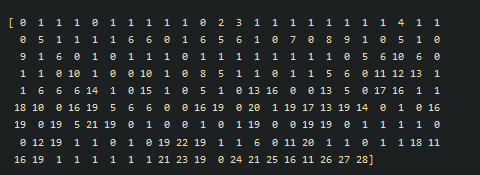
**2.数据预处理**初始数据共有四列：日期，天气状况，气温，风力风向

日期：提取月日，转换为int，因为根据经验气候显然与四季日期有关

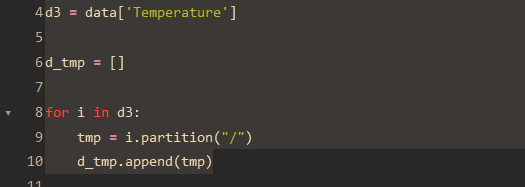


天气状况：分析课知可能情况不多，直接使用astype函数转换为int

****

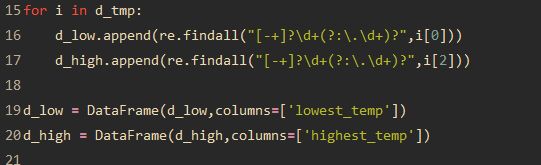
****

气温：该属性包含高温和低温，使用正则划分



可得到当天最高温度和最低温度。

再使用正则表达式仅提取温度中的数字去掉单位



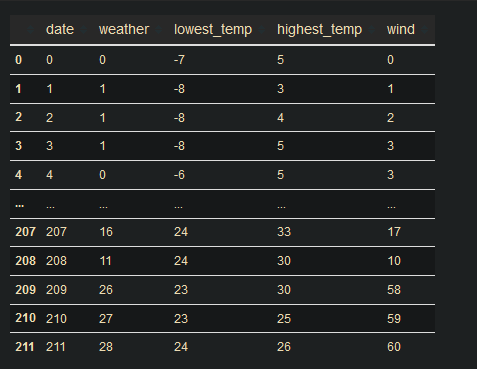
风力风向：

同样使用astype函数转换成数字方便后续回归。

将这些数据整合存为data.xlsx方便后续使用

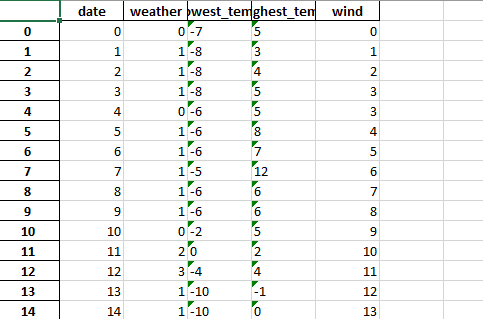








数据如下图



**3.线性回归**

通过梯度下降法，找到代价函数最小的参数求最值，其中注意步长的选取，不能过大忽略极值也不能过小使得迭代次数过多

Code：

def normalization(dataX):

'''z-score归一化

Parameters:

dataX: 输入特征向量

Returns:

dataX1: 归一化之后的dataX

mu: 均值向量

sigma: 标准差向量

'''

mu = dataX.mean(0)

sigma = dataX.std(0)

dataX1 = (dataX - mu) / sigma

return dataX1, mu, sigma

class linear\_regression():

def \_\_init\_\_(self, dataX, dataY, alpha=0.1):

ones = np.ones((dataX.shape[0], 1))

self.dataX = np.c\_[ones, dataX]

self.dataY = dataY

self.datasize = self.dataX.shape[0]

self.alpha = alpha # 学习率

self.theta = np.zeros(self.dataX.shape[1]) # 参数初始化

def fit(self):

iterations = 100 # 设置停止迭代规则，迭代100次就停止更新

for i in range(iterations):

y = np.dot(self.dataX, self.theta).reshape(self.datasize,) # 数据集矩阵与参数向量相乘，得到一个向量，为各个样本的预测值

# print(y)

loss = 1 / self.datasize \* sum((y - self.dataY) \*\* 2) # 计算平方差损失

print("第", i, "次迭代，损失为：", loss)

# 更新theta值

updated\_theta = []

for j, theta\_j in enumerate(self.theta):

x\_j = self.dataX[:, j]

updated\_theta\_j = theta\_j - self.alpha \* (1 / self.datasize) \* np.sum((y - self.dataY) \* x\_j)

updated\_theta.append(updated\_theta\_j)

self.theta = np.array(updated\_theta)

print("训练完成！")

def predict(self, testX):

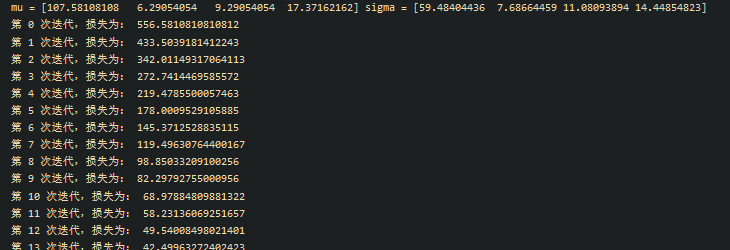
ones = np.ones((testX.shape[0], 1))

testX = np.c\_[ones, testX]

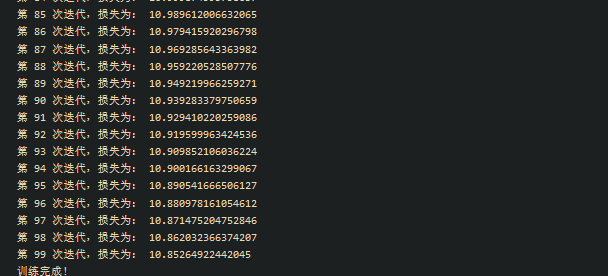
y\_hat = np.dot(testX, self.theta)

return y\_hat

迭代一百次：



……



现在得到了一个线性回归的模型。

在选取评价指标时，因为回归预测，所以选取了MSE，MAE，RMSE，R2四个指标进行评价。

同时这四个指标也是线性回归最为常见的指标。

MSE：真实值-预测值 然后平方之后求和平均。（同时为线性回归损失函数）

MAE：平均绝对误差

RMSE：是MSE开根号的结果，保持数据在同一数量级

R2：如果结果是0，就说明我们的模型很差。如果结果是1。就说明我们模型无错误。如果结果是0-1之间的数，就是我们模型的好坏程度。

实现代码如下：

#输出回归问题的评价指标

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

mean\_squared\_error(y\_test, y\_predict)

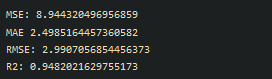
MSE = mean\_squared\_error(y\_test,y\_predict)

MAE = mean\_absolute\_error(y\_test,y\_predict)

RMSE = np.sqrt(mean\_squared\_error(y\_test,y\_predict))

R2 = r2\_score(y\_test,y\_predict)

具体结果如图所示



可以看出绝对误差较小，模型预测很成功

**4.决策树**决策树（Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。我们在id3和c4.5的基础上深入学习，实现了一个cart决策树。对天气进行分类。

分类问题常用的评价指标有四种，分别为精确度，准确度，f1，召回率，使用准确度和精确度进行测试。

Code:

class CART\_DecisionTree(object):

"""

基于cart算法的决策树

"""

def \_\_init\_\_(self, \_max\_depth, \_min\_splits):

self.max\_depth = \_max\_depth # 最大的深度

self.min\_splits = \_min\_splits # 最小的叶子节点数

def fit(self, \_feature, \_label):

self.feature = \_feature

self.label = \_label

self.train\_data = np.column\_stack((self.feature,self.label))

self.build\_tree() # 构造二叉树

def compute\_gini\_similarity(self, groups, class\_labels):

"""

计算基尼系数

"""

num\_sample = sum([len(group) for group in groups]) # 计算总数目

gini\_score = 0

# 对在该划分下求基尼系数

for group in groups:

size = float(len(group))

if size == 0:

continue

score = 0.0

for label in class\_labels:

porportion = (group[:,-1] == label).sum() / size

score += porportion \* porportion

gini\_score += (1.0 - score) \* (size/num\_sample)

return gini\_score

def terminal\_node(self, \_group):

"""

获取叶子节点中的最多数量的类的类别作为分类类别

"""

class\_labels, count = np.unique(\_group[:,-1], return\_counts= True)

return class\_labels[np.argmax(count)]

def split(self, index, val, data):

"""

根据值划分成两组，分组过程,其中蕴含排序

"""

data\_left = np.array([]).reshape(0,self.train\_data.shape[1])

data\_right = np.array([]).reshape(0, self.train\_data.shape[1])

# 进行划分 小于等于的val的在data\_left，大于val的在data\_right

for row in data:

if row[index] <= val :

data\_left = np.vstack((data\_left,row))

if row[index] > val:

data\_right = np.vstack((data\_right, row))

return data\_left, data\_right

def best\_split(self, data):

"""

找到最优切分点

"""

class\_labels = np.unique(data[:,-1]) # 获取类别

best\_index = 999

best\_val = 999

best\_score = 999

best\_groups = None

for idx in range(data.shape[1]-1):

for row in data:

groups = self.split(idx, row[idx], data)

gini\_score = self.compute\_gini\_similarity(groups,class\_labels) # 得到当前划分下的基尼系数

if gini\_score < best\_score:

# 小于当前的最小基尼系数的时候更新

best\_index = idx

best\_val = row[idx]

best\_score = gini\_score

best\_groups = groups

result = {}

result['index'] = best\_index

result['val'] = best\_val

result['groups'] = best\_groups

return result

def split\_branch(self, node, depth):

"""

以递归的方式分枝，直到满足对最大深度或者小于最小的叶节点数目

"""

left\_node , right\_node = node['groups']

del(node['groups']) # 删除这一key

if depth >= self.max\_depth:

# 超过深度

node['left'] = self.terminal\_node(left\_node)

node['right'] = self.terminal\_node(right\_node)

return None

if len(left\_node) <= self.min\_splits:

# 小于最小节点数量的时候

node['left'] = self.terminal\_node(left\_node)

else:

node['left'] = self.best\_split(left\_node)

# 递归

self.split\_branch(node['left'],depth + 1)

if len(right\_node) <= self.min\_splits:

node['right'] = self.terminal\_node(right\_node)

else:

node['right'] = self.best\_split(right\_node)

# 递归

self.split\_branch(node['right'],depth + 1)

def build\_tree(self):

"""

递归的方式创建树

"""

self.root = self.best\_split(self.train\_data)

self.split\_branch(self.root, 1)

return self.root

def \_predict(self, node, row):

"""

通过递归的方式查找树来预测

"""

if row[node['index']] < node['val']:

if isinstance(node['left'], dict):

return self.\_predict(node['left'], row)

else:

return node['left']

else:

if isinstance(node['right'],dict):

return self.\_predict(node['right'],row)

else:

return node['right']

def predict(self, test\_data):

"""

返回的是概率

"""

self.predicted\_label = np.array([])

for idx in test\_data:

self.predicted\_label = np.append(self.predicted\_label, self.\_predict(self.root,idx))

return self.predicted\_label

def accuracy(self,x\_test,y\_test):

y\_predict = self.predict(x\_test)

return np.sum(y\_test==y\_predict)/y\_test.shape[0]

def macro\_precision(self,x\_test,y\_test):

y\_unique = np.unique(y\_test)

y\_predict = self.predict(x\_test)

P = []

for i in y\_unique:

P.append(np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_predict==i))

return np.sum(P)/len(P)

def macro\_recall(self,x\_test,y\_test):

y\_unique = np.unique(y\_test)

y\_predict = self.predict(x\_test)

R = []

for i in y\_unique:

R.append(np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_test==i))

return np.sum(R) / len(R)

def macro\_f1\_score(self,x\_test,y\_test):

y\_unique = np.unique(y\_test)

y\_predict = self.predict(x\_test)

F = []

for i in y\_unique:

p = np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_predict==i)

r = np.sum(y\_predict[y\_predict==y\_test]==i)/np.sum(y\_test==i)

F.append((2\*p\*r)/(p+r))

return np.sum(F) / len(F)

def precision(self,x\_test,y\_test):

y\_predict = self.predict(x\_test)

y\_unique = np.unique(y\_test)

return np.sum(y\_predict[y\_predict==y\_test]==y\_unique[0])/np.sum(y\_predict==y\_unique[0]),np.sum(y\_predict[y\_predict==y\_test]==y\_unique[1])/np.sum(y\_predict==y\_unique[1])

def recall(self,x\_test,y\_test):

y\_predict = self.predict(x\_test)

y\_unique = np.unique(y\_test)

return np.sum(y\_predict[y\_predict==y\_test]==y\_unique[0])/np.sum(y\_test==y\_unique[0]),np.sum(y\_predict[y\_predict==y\_test]==y\_unique[1])/np.sum(y\_test==y\_unique[1])

def f1\_score(self,x\_test,y\_test):

precision = self.precision(x\_test,y\_test)

recall = self.recall(x\_test,y\_test)

return 2\*precision[0]\*recall[0]/(precision[0]+recall[0]),2\*precision[1]\*recall[1]/(precision[1]+recall[1])

def report(self,x\_test,y\_test,labels):

y\_predict = self.predict(x\_test)

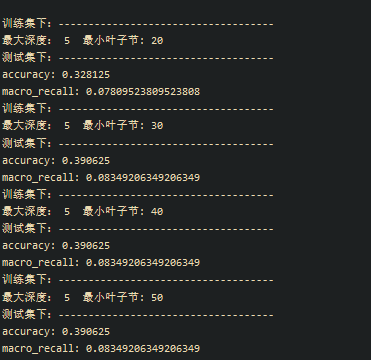
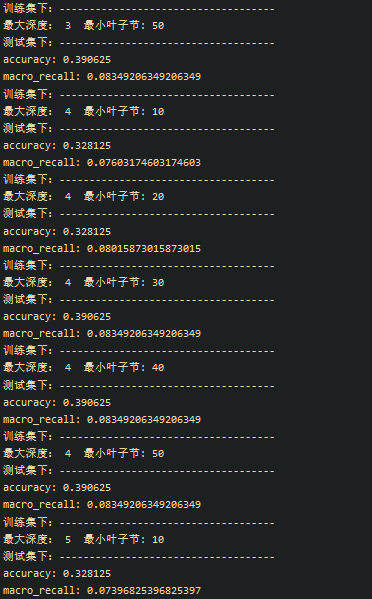
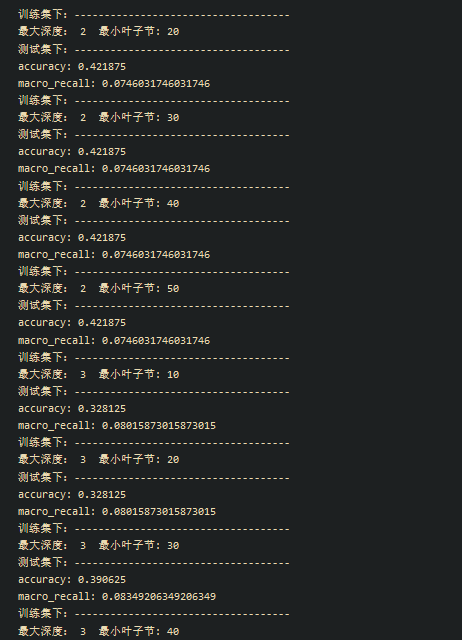
y\_predict = y\_predict.astype(y\_test.dtype)

report = classification\_report(y\_test,y\_predict,labels=labels)

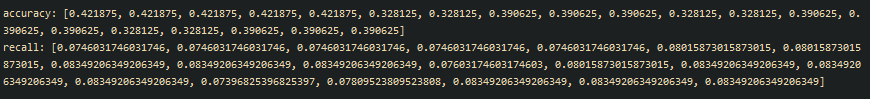
return report

选择深度二到五，叶子节点20，30，40，50分别进行分类

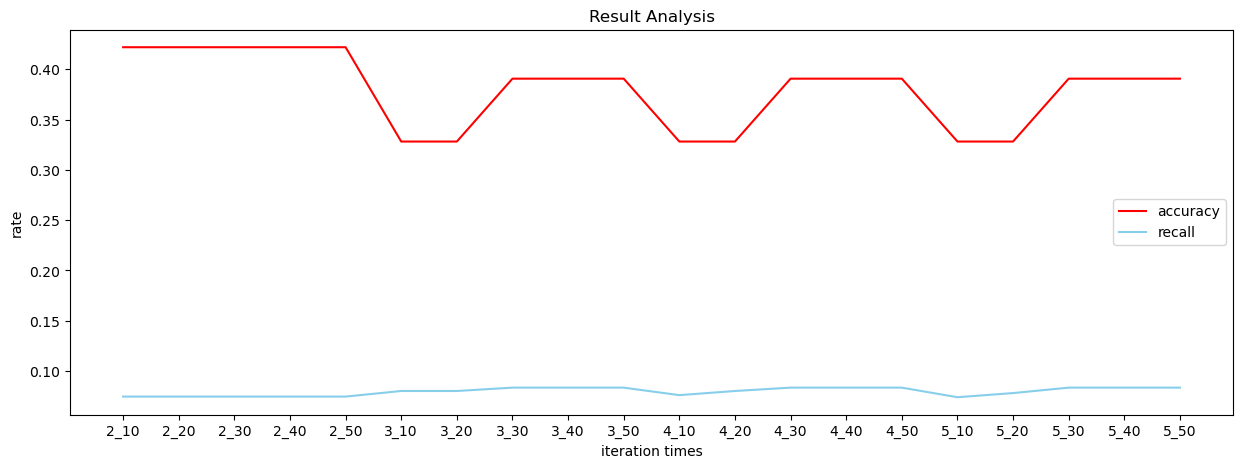
过程如图



两种指标统计统计



作图



可以看出，最小叶子节点在10~20之间时，accuracy效果最坏，但总能保持一个不错的水平，分类数量在2到5时对召回率影响不大，考虑之前对于数据的分析，应该和全部天气的可能情况接近时效果最好。

# 五、实验总结

在数据获取中，使用了爬虫获得自己数据，在数据预处理中，进行缺失值处理和归一化处理；在特征工程中，根据特征相关性进行特征选择提取等；在模型选择和调参中，选用梯度下降线性回归和CART决策树两个模型进行对比并选择随机森林算法和模型结构，对模型超参数进行调整；在模型评价中，选择了回归和分类的常见指标对模型性能进行评估

# 六、感想及体会

**学习爬虫的体会：**

1. **数据发现与探索**：爬虫让我深入了解网页背后的数据世界，学到了如何发现和获取数据，以及如何从大量的信息中提炼出有价值的内容。
2. **技术挑战与解决问题**：爬虫开发可能面临各种技术挑战，比如处理不同结构的网页、处理动态加载的内容、应对反爬虫机制等。解决这些问题会提高技术能力和解决问题的能力。
3. **合规与道德**：学习爬虫也需要了解数据采集的合规性和道德规范。在获取数据时，学到了需要遵守网站的规定和法律法规，确保合法合规。
4. **数据处理与分析**：爬取到的数据通常需要处理和分析。学习爬虫的过程也会接触数据处理、清洗、分析和可视化的技能。
5. **自动化与效率**：爬虫技术能够自动化数据收集过程，提高数据获取的效率。这种自动化的思维方式也能够在其他领域带来启发。
6. **持续学习**：由于网络环境不断变化，爬虫技术也在不断发展。学习爬虫是一个持续学习的过程，需要不断跟进新技术和方法。

**学习数据预处理的感悟**

1. **数据的多样性和复杂性**：数据预处理让我意识到真实世界中的数据通常是非常复杂和多样的。数据可能来自不同的来源，有着各种各样的格式和结构，需要经过清洗和转换才能变得有用。
2. **数据质量的重要性**：数据质量对分析和建模的影响极大。学习数据预处理会让你明白数据的质量对最终的分析结果产生重要影响，而清洗和处理数据是确保数据质量的关键一步。
3. **特征工程的重要性**：数据预处理中的特征工程是提取、创建和选择最相关特征的过程。它能够极大地影响到机器学习模型的性能，因此需要仔细考虑特征的选择和转换。
4. **数据可视化的力量**：在数据预处理过程中，数据可视化是一个重要的步骤，能够帮助你更好地理解数据。可视化让复杂的数据变得更加直观和易于理解。
5. **处理缺失值和异常值**：学习数据预处理让我了解到数据中常见的问题，如缺失值和异常值的处理。这些问题的处理能够提高数据的准确性和可靠性。
6. **算法和模型的需求**：数据预处理是为了让数据更适合于机器学习算法和模型的需求。了解这些需求有助于后续更好地选择合适的预处理方法。

**学习基于梯度下降的线性回归感悟**

1. **数据与模型的关系**：学习回归让我深入了解数据和模型之间的关系。通过建立模型预测连续数值，你能够更好地理解数据中变量之间的关系和趋势。
2. **模型的拟合与预测**：回归模型的目标是找到最能描述数据的函数。学习如何拟合数据，并使用模型进行预测，意识到模型的适应能力和泛化能力之间的平衡。
3. **特征选择和工程**：回归问题中，选择合适的特征对模型性能至关重要。学习回归模型可以了解到如何选择和转换特征以提高模型的预测能力。
4. **误差分析与改进**：在回归模型中，误差分析能够帮助理解模型预测的准确性。通过分析模型的误差能够确定改进模型的方向。
5. **模型的评估与优化**：学习回归模型也包括了模型评估和优化的过程。了解不同评估指标和优化方法对模型性能的影响，有助于构建更准确的模型。
6. **对数据背后规律的理解**：回归模型的学习让我更深入地理解数据背后的规律和趋势，这有助于预测和决策。

**学习CART决策树的感悟**

1. **树形结构的直观性**：CART 决策树以树形结构展现数据的分类或回归过程，这种结构直观、易于理解。通过树的分支和节点，能够清晰地了解决策过程。
2. **特征选择的重要性**：学习 CART 决策树让我了解到在每个节点选择最佳特征的重要性。提醒了我特征选择对模型构建的影响，以及如何根据特征划分数据集以提高预测准确性。
3. **对数据的划分和分类**：决策树基于数据特征进行划分，这种根据特征进行分类的方式让我更深入地了解数据之间的关系和区别。
4. **剪枝与泛化能力**：学习 CART 决策树时，你会了解到过拟合问题以及如何通过剪枝提高模型的泛化能力。这让我明白了模型复杂度和模型性能之间的平衡。
5. **解释性和可视化**：决策树模型具有很强的解释性，能够清晰地解释模型的预测过程。同时，通过可视化决策树，能够直观地展示决策的逻辑和分支。
6. **集成学习的基础**：CART 决策树常常作为集成学习中弱学习器的基础，学习 CART 决策树对于理解集成学习算法（如随机森林）有着重要的启发作用。