**大数据生态学研究方法-笔记-SA23008216**

1. **生态学研究及生态学范式**

**2024.3.1**

旧范式的假设检验的逻辑是：针对某个科学问题，构思一个假说（有关变量），假设变量之间存在因果关系，然后根据假说推演可验证的预言，再设计实验，收集数据，验证预言，最后依据验证结果，拒绝或接受假说。

①二分思维：传统假设检验是基于二元逻辑思维（接受或拒绝零假设），忽略了中间地带的信息，不能提供全面的分析。

Fisher的证伪主义：其假设检验的基本思想是小概率反证法，即构造一个小概率事件或原假说(H0)，同时提出另一个与原假说对立的备择假说(H1)。根据这个思想设计实验，即从总体中抽出样本，依据样本计算统计量(t、x2或F)的值，根据预设的概率水平(p<0.01或p<0.05)检验和选择的统计量的概率分布，确定原假说成立的可能性大小，做出拒绝或接受H0判断。

Fisher主张的H0假说是小概率事件，小概率事件在一次实验中不可能发生。但生态过程不断演化，无法预见原假说是小概率事件。因此，生态学的所有试验验证逻辑上应倾向于证实假说(H0)成立。

②假设条件限制：传统假设检验通常需要满足一些假设条件，如正态分布、独立性等，而现实数据往往不完全符合这些假设条件，导致结果的可靠性受到影响。

与物理、化学等学科不同，生态学问题的答案不是简单的“是”或“不是”，通过H0显著性检验来验证生态学假说并不严谨。生态学中本身可能无法确定H0的初始条件，甚至可能不存在H0，构建单一主导因子的H0很难。例如检验群落中竞争的作用，这无法通过一个简单实验进行证伪，因为群落中同时存在竞争、捕食和寄生、干扰等多因素作用。

③操控实验：操控实验包括正交实验设计法与析因法，无论哪种实验都需要遵循重复、随机化、区组化的原则。但是由于研究对象所处的环境是开放的，其中影响因子及相互关系难以被发现和确定。另外，在大尺度环境下，要采取严谨和一致操作才能排除无关干扰。

但生态学实验研究过程中，部分研究对象研究尺度过大，研究时间过长，在此之中，生态过程在不断演化，生态规律也在不断变化，很难进行实验的重复和一致。

④过度依赖于显著性水平：传统的假设检验通常只关注结果是否显著，而忽略了效应的大小和实际意义。这可能导致对结果的片面解释。

⑤样本量要求：传统假设检验通常对样本量有一定的要求，特别是在小效应或复杂模型下，可能需要大量样本才能得到可靠的结果。

⑥多重比较问题：在进行多个假设检验时，存在多重比较问题，容易导致统计推断的偏差，需要采取措施来控制错误发现率。

①相关性分析：科学研究的目的是寻找现象之间的因果关系。

分析变量之间的相关性比探寻因果更重要，基于相关分析的预测是大数据研究的核心。按照科学哲学观点，如果一个命题能够解释以往出现的现象，又能预测未来可能出现的新现象和新问题，其科学性就得到了检验。一方面数据规律本身是从过去所积累的数据中挖掘出来的，完全可解释过去的现象或问题，另一方面，大数据包含了海量的各种现实数据，通过机器学习过去的经验来推测未来。因此，与因果律相比，基于大数据相关性的预测更准确，而且不易受偏见的影响。

②全数据归纳：样本≠总体，避免经验科学范式≠因果。

数据科学范式科学发现的逻辑起点不同于其他范式，主要表现在如下三个方面：第一，经验科学范式认为“科学始于观察”，即在自然和实验观察的基础上，通过归纳提炼出科学理论；假说-验证范式主张科学发现始于科学问题；而数据科学颠覆了原来的科学发现模式，从数据出发，利用数据挖掘方法发现数据中蕴含的规律性，形成了“科学始于数据”的新模式。

③利用非参数或半参数模型。

生态学研究强调生物与环境之间的关系，因此确定影响某个生态过程或现象的关键因子是首要任务。基于传统的统计分析方，探索数据分布特征，结合决策树、随机森林等分析特征的相关性，计算特征重要性，就可筛选出关键特征。利用提升回归树、朴素贝叶斯等建立分类与回归模型，用于生态预测或预警。基于关键特征，利用聚类分析方法，发现如空间上的hotspots、时间上的Motifs等模式。

④将数据科学范式与理论科学范式融合，避免归纳偏见。

数据科学采用归纳方法，而不是演绎逻辑。经验科学采用的是不完全归纳。数据科学范式沿袭了经验科学的归纳逻辑，不同的是数据科学采用的是全数据模式，即“样本=整体”的完全归纳法，克服了小样本不完归纳法的局限性，利于发现异常值。

1. **R语言及相关包、函数**
   1. **安装R及Rstudio、将Rstudio与Github相连接**

**2024.3.6-2024.3.8**

Rstudio：打开RStudio→“Tools”、“Global Options”→“Git/SVN”选项卡→“Enable version control interface for RStudio”→“Git”→单击“Create RSA Key”生成新的SSH密钥→选择默认路径并单击“Create”

Github：登录账户→“Settings”→“SSH and GPG keys”→“New SSH key”→“Key”字段中粘贴RStudio生成的公钥→“Add SSH key”，将公钥添加到GitHub账户中→

返回RStudio，并单击“commit”按钮，以确保RStudio可以成功连接到GitHub。

* 1. **R的函数及程序包**

1. 基本数学运算

①**算术运算符**：- 、+ 、\*、/、^、、Log（）、sqrt（），减、加、乘、除、指数、对数、平方根

②**模运算符**：%%，运算符前面的值除以运算符后面的值，并返回两个值相除的余数

③**矩阵乘法符号**：%\*%，前后均为矩阵，返回一个矩阵，为前后两矩阵的乘积

④**关系、逻辑运算符**：<、>、=、！=、>=、<=、|、&，小于、大于、等于、不等于、大于等于、小于等于、或、与

⑤**赋值操作符**：<- 、->，将数值或对象赋给变量 eg：x <- 5，将5赋给变量 x

⑥**rep函数**:复制字符串或数值

eg：rep("words", 4) #复制 "words" 字符串4次

⑦**Seq函数**：使用seq函数创建数值序列

eg：seq(1, 5, 1) #创建从1到5的序列，步长为1

1. 函数

①检验函数：t-test（）、var.test（）

②绘图函数：plot(), lines(), legend(), abline()

参数设置：pch, cex, col, type, lwd, lty, main, xlab, ylab

eg：

plot(x, y): 使用给定的x和y值创建散点图。

pch: 指定点的形状。0表示方块，1表示圆圈，3表示三角形，4表示十字，以此类推。

cex、col: 指定点的大小、颜色。

type: 指定绘制类型。"l"表示连接线，"o"表示连接线和点，"p"表示只绘制点，"b"表示绘制点并连接线。

lty: 指定线的类型。例如，1 表示实线，2 表示虚线，以此类推。

Lwd、main、col.main: 指定线的宽度、图的标题、标题的颜色。

xlab 和 ylab: 指定 x 轴和 y 轴的标签。

legend(): 添加图例，指定位置、标签、线型、颜色等信息。

lines(): 添加额外的线到已有的图中。

abline(): 添加直线到图中，可设置截距和斜率。

③其他函数

colors(): 查看可用颜色

?plot.function: 查询函数帮助文档

1. R包

①R包安装与加载

install.packages() #安装R包

library() #加载R包

devtools::install\_github() #从GitHub安装R包

Repos # 指定特定仓库安装R包

type='source' # 指定安装源为源代码

②使用pak安装R包

pak::pkg\_install() # 使用pak包安装R包

③获取帮助文档和演示

help(package="packageName") #查看R包帮助文档

vignette() #查看R包演示文档

browseVignettes() #浏览R包演示文档

demo(package="packageName") #运行R包的演示

④搜索帮助

apropos("^keyword") #搜索包含指定关键字的函数

ls("package:packageName") #列出指定包中的所有函数

help.search("^keyword") #在帮助文档中搜索指定关键字

⑤自定义函数

创建函数：使用function()关键字

调用函数：直接使用函数名并提供参数

source() #执行外部R脚本文件中的函数

⑥RStudio环境配置

usethis::edit\_r\_environ() #编辑R环境文件

Sys.getenv("variable") #获取环境变量的值

⑦其他

pak::pak() #使用pak包进行包管理

gptstudio:::gptstudio\_chat() #使用gptstudio包进行聊天交互安装R及Rstudio

rm(list = ls()) #清空当前环境中的所有变量

cat("\014") #清屏控制台输出

print() #输出变量值

cat() #输出变量值和其他字符串，用于拼接输出

1. **Rstudio操作相关（数据类型、简单运算...）**
2. 读取与保存函数：read.csv() #读取矩阵或表类型数据；

write() #保存数据；

write.xlsx() #将数据框保存为Excel文件；

read.xlsx() #从Excel文件读取数据；

write.csv() #将数据框保存为CSV文件；

read.csv() #从CSV文件读取数据。

1. 数据类型

①**向量vector**：数字向量（储存整数或者小数）、字符向量（储存字符串）

eg:a<-c(1:10) #创建1~10的数字向量；

a<-“hi” #创建字符向量。

②**矩阵（matrix）**：保存多维数据，matrix（）函数创建矩阵，二维数组，所有元素必须是相同的数据类型。

eg：a<-matrix（c（1：10），nrow=2，ncol=5，byrow=FALSE） #由a（）创建一个1到10的向量，按列从左到右，从上到下排列2×5的矩阵。

注：dim可查看矩阵维度，也可把向量转换为矩阵。

③**列表**：其他对象的集合，可同时包含字符向量、数字向量和一个矩阵。List（）函数创建列表。

④**数据框**：带有data.frame的列表，允许多种数据类型放在一起（向量、矩阵、列表或其他），但添加到数据框的组件需要有相同的行数。data.frame()函数创建数据框。

注：str（）函数可查看数据框的结构。

⑥**数组（array）**：多维数据结构，可以包含相同类型的元素。

1. 其他

①索引取值：索引运算符：[ ]，可从向量、列表、矩阵或数据框中提取单个元素。

使用方括号[ ]选择列，结果为数据框;使用$符号选择列，结果为向量;[ ]和$可以用于选择数据框中的列，结果为向量。

eg：a<-c(1,2,3,4,5,6,7) #创建向量

a<-[3] #从向量a中提取第三个元素

a<-[-3] #从向量a中删除第三个元素

a[2:4] #从向量a中提取第二到四个元素

b=matrix（c（1：9），nrow=3,ncol=3） #创建矩阵

b[2,] #从矩阵中提取第二行

b[,2] #从矩阵中提取第二列

②数据转换

as.data.frame() #将矩阵转换为数据框

data.matrix() #将数据框转换为矩阵

array() #创建数组

unlist() #将数据框中的列转换为向量

factor() #将向量转换为因子

levels() #查看因子的水平

nlevels() #查看因子的水平数量

eg:a<-matrix（c（1：10），nrow=2，ncol=5，byrow=FALSE） #创建矩阵

Class(a) #返回a的类别

df<-as.data.frame(a) #转换为数据框

class(df)

V<-as.vector(a) #转换为向量

Class(V)

List<-as.list(a) #转换为列表

Class（List）

③apply家族

apply()函数用于在矩阵的行或列上应用指定的函数;

lapply()函数用于对列表中的每个元素应用指定的函数;

sapply()函数与lapply()类似，但返回的是向量而不是列表;

tapply()函数根据指定的因子对向量进行分组，并对每个组应用指定的函数。

④ tidyverse包

select() #选择数据框中的列，通过列名、位置、范围或条件选择列。

Eg:select(dataframe, col1, col2) # 选择指定列

select(dataframe, starts\_with("col")) # 选择以 "col" 开头的列

select(dataframe, -col\_to\_exclude) # 排除指定列

filter() #筛选数据框中的行，使用逻辑条件筛选行。

filter(dataframe, condition) # 筛选符合条件的行

filter(dataframe, col1 > 10 & col2 == "value") # 多条件筛选

mutate() #在数据框中添加新列或修改现有列，基于现有列进行计算并创建新列。

mutate(dataframe, new\_col = col1 + col2) # 添加新列并计算

mutate(dataframe, col1\_squared = col1^2) # 计算现有列的平方并添加新列

group\_by() #按照指定的列对数据框进行分组

summarize() #用于对数据进行汇总统计

gather() #将宽格式数据框转换为长格式。将列名转换为两个新的列：一个用于存储原始列名的值，另一个用于存储原始列中的值。

spread() #将长格式数据框转换为宽格式。将两列的值转换为一个新的列，其中一列用作新列的名称，另一列用作新列的值。

⑤其他

as.Date() #将字符向量转换为日期对象

sum() #计算向量或数组的总和

paste() #拼接字符串

ymd() #将字符型日期转换为Date对象

class() #返回对象的类别

typeof() #返回对象的类型

plotKM:用于制作KML文件的函数包。

lubridate包:用于处理日期和时间数据。

1. **树模型、caret包（重点：随机森林模型）**

①树模型：决策树、随机森林、提升回归树

决策树（Decision Trees）:决策树是一种基于树结构的机器学习模型，用于解决分类和回归问题。它通过对数据集的属性进行递归划分，构建一系列的决策规则，以达到对目标变量的准确预测。决策树的构建过程如下：

特征选择：根据某种准则（如信息增益、基尼不纯度或方差减少）选择最佳的特征来进行节点的分裂。

节点分裂：将数据集根据选定的特征划分为子集，每个子集对应于特征的一个取值或一个范围。

递归：对每个子集重复以上步骤，直到满足停止条件（如达到最大深度、节点样本数小于阈值等）。

决策树的优点包括易于理解和解释、能够处理数值型和类别型数据、不需要对数据进行归一化或标准化。然而，决策树容易过拟合，并且对数据中的噪声和缺失值敏感。

随机森林（Random Forest）:随机森林是一种基于集成学习的算法，它通过构建多个决策树来提高预测性能。随机森林的关键特点包括：

随机选择特征：在每棵决策树的构建过程中，随机森林会从原始特征集合中随机选择一部分特征用于节点分裂，以增加模型的多样性。

投票或平均：对于分类问题，随机森林会将所有决策树的预测结果进行投票，选择得票最多的类别作为最终预测结果；对于回归问题，则取所有决策树的预测结果的平均值。

Bagging：随机森林使用了自助采样法（Bootstrap Aggregating，简称Bagging）来训练每个决策树，即通过有放回地随机抽样生成多个不同的训练集。

随机森林具有较高的预测准确度、对噪声数据具有较好的鲁棒性、不容易过拟合等优点。它在解决各种分类和回归问题上都表现出色。

提升回归树（Gradient Boosted Regression Trees，GBRT）:提升回归树是一种集成学习方法，它通过逐步构建多个决策树来提高模型性能，每棵树都是在之前树的残差上进行拟合。提升回归树的主要特点包括：

顺序构建：每棵树都是串行构建的，即每一棵树都在之前树的基础上进行构建。

残差拟合：在每一轮迭代中，新的树都会针对之前模型的残差进行拟合，以逐步减小残差，从而提高模型的拟合效果。

Shrinkage：为了防止过拟合，通常会引入学习率（learning rate）参数，用于对每棵树的贡献进行缩减。

**①回归树模型示例:**

# 加载所需的库

library(rpart) # 用于构建决策树模型

library(rpart.plot) # 用于绘制决策树图

library(fastDummies) # 用于创建虚拟变量

library(randomForest) # 用于构建随机森林模型

library(caret) # 用于计算特征重要性和评估模型

library(Metrics) # 用于计算均方根误差

# 从CSV文件中加载数据

data <- read.csv("data/MLdata/dickcissel.csv", stringsAsFactors = TRUE)

# 删除含有缺失值的观测

df <- na.omit(data)

# 查看数据前几行

head(df)

# 预处理数据

# 标准化除目标变量外的所有数值型特征

df[3:ncol(data)] <- scale(df[3:ncol(df)])

# 将分类变量转换为虚拟（二进制）变量

df <- fastDummies::dummy\_cols(df, remove\_first\_dummy = TRUE, remove\_selected\_columns = TRUE)

# 将数据集分割成训练集和测试集

trainingRowIndex <- sample(1:nrow(df), 0.75 \* nrow(df))

dtrain <- df[trainingRowIndex, ] # 模型训练数据

dtest <- df[-trainingRowIndex, ] # 测试数据

# 创建决策树模型

dt\_tree <- rpart(abund ~ ., data = dtrain, method = 'anova')

# 可视化决策树模型

rpart.plot(dt\_tree, box.palette = "RdBu", shadow.col = "gray", nn = TRUE)

# 计算特征重要性

varimp.dt\_tree <- varImp(dt\_tree)

# 使用测试数据集评估模型

dtest$dt\_tree.pred <- predict(dt\_tree, newdata = dtest)

# 查看预测结果和实际值

head(dtest[c("abund", "dt\_tree.pred")], n = nrow(dtest))

# 计算均方根误差

rmse\_dt <- Metrics::rmse(dtest$abund, dtest$dt\_tree.pred)

cat("决策树模型的均方根误差:", rmse\_dt, "\n")

**②随机森林回归模型的示例：**

# 构建随机森林回归模型

rf\_tree <- randomForest(abund ~ ., data = dtrain, proximity = TRUE, ntree = 1000)

# 计算特征重要性

varimp.rf\_tree <- caret::varImp(rf\_tree)

# 使用测试数据集评估随机森林回归模型

dtest$rf\_tree.pred <- predict(rf\_tree, newdata = dtest)

# 计算均方根误差

rmse\_rf <- Metrics::rmse(dtest$abund, dtest$rf\_tree.pred)

cat("随机森林回归模型的均方根误差:", rmse\_rf, "\n")

**③使用caret包在不同算法下训练模型，并比较它们的性能**

# 定义控制参数

ctrl <- trainControl(method = "repeatedcv", # 使用重复交叉验证

repeats = 3, # 重复3次

number = 5) # 5折交叉验证

# 定义要比较的算法

algorithms <- c("rpart", "rf", "gbm") # 决策树、随机森林和GBM

# 使用caret包训练模型

models <- caret::train(abund ~ ., data = dtrain,

method = algorithms,

trControl = ctrl)

# 输出模型性能比较结果

print(models)

# 可视化模型性能比较结果

plot(models)

**④用R中的caret包来同时构建决策树、随机森林和梯度提升树（GBM）三种算法的回归模型，并对它们的性能进行比较。**

加载必要的库和数据：

library(caret)

library(Metrics)

# 从CSV文件加载数据

data <- read.csv("data/MLdata/dickcissel.csv", stringsAsFactors = TRUE)

# 将数据分为训练集和测试集

set.seed(123)

trainIndex <- createDataPartition(data$abund, p = 0.7, list = FALSE, times = 1)

data\_train <- data[trainIndex,]

data\_test <- data[-trainIndex,]

定义交叉验证和预处理参数：

# 定义交叉验证方法和重复次数

fitControl <- trainControl(method = "repeatedcv", number = 5, repeats = 2)

# 定义预处理参数

preProcOptions <- c('scale', 'center')

训练决策树模型：

model\_rpart <- train(abund ~ ., data = data\_train, method = "rpart",

trControl = fitControl, preProcess = preProcOptions,

tuneLength = 5, metric = "RMSE")

进行预测并计算模型性能：

predictions\_rpart <- predict(model\_rpart, newdata = data\_test)

rmse\_rpart <- rmse(data\_test$abund, predictions\_rpart)

训练随机森林模型：

model\_rf <- train(abund ~ ., data = data\_train, method = "rf",

trControl = fitControl, preProcess = preProcOptions,

tuneLength = 5, metric = "RMSE")

训练梯度提升树（GBM）模型：

model\_gbm <- train(abund ~ ., data = data\_train, method = "gbm",

trControl = fitControl, preProcess = preProcOptions,

tuneLength = 5, metric = "RMSE")

比较模型性能并绘制箱线图：

models\_compare <- resamples(list(TREE = model\_rpart, RF = model\_rf, GBM = model\_gbm))

summary(models\_compare)

bwplot(models\_compare, scales = list(x = list(relation = "free"), y = list(relation = "free")))

**5.安装并配置 R 与 Python 的交互环境**

install.packages('reticulate') #安装R 中用于与 Python 交互的 reticulate 包

reticulate::install\_miniconda() #安装Miniconda，用于管理 Python 环境

reticulate::py\_install('retriever') #安装Python中retriever包，用于数据检索

install.packages('rdataretriever') #安装R中rdataretriever包，此包提供了用于运行检索器的接口

dataretriever::get\_updates() #更新了可用的数据集列表