往期回顾

前言

线性回归一般是用来预测连续因变量（目标变量）的模型，同时，它也可以用来选择核心变量（即真正影响因变量的自变量有哪些）。关于如何构建并求解多元线性回归模型的理论部分我们已经在《》中做了详细的梳理，包括模型的偏回归系数的计算、模型的显著性检验和偏回归系数的检验。如果你对理论部分还不是很明白的，建议你先看一下我之前写的文章。

在本期的推文中，我们将手把手的分享如何使用Python和R语言实现多元线性回归模型的落地。如果你对这篇文章感兴趣，希望能够看完下面的内容，相信对你有一定的帮助，同时，文末部分也会给出相关脚本和数据集的下载链接。

案例分享

销售额与广告渠道的关系

如果市场的运营部门给了你一份数据，数据包含了不同广告渠道的成本及对应的产品销售量。现在的问题是：

- 哪些渠道的广告真正影响了销售量？
- 根据已知的渠道预算，如何实现销售量的预测？
- 模型预测的好坏，该如何评估？

利用Python建模

哪些渠道的广告真正影响了销售量？对于这个问题的回答，其实就是在构建多元线性回归模型后，需要对偏回归系数进行显著性检验，把那些显著的变量保留下来，即可以认为这些变量对销售量是存在影响的。关于线性回归模型的落地，我们这里推荐使用statsmodels模块，因为该模块相比于sklearn，可以得到更多关于模型的详细信息

```
# ======= Python3 + Jupyter =======
# 导入第三方包
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# 读取外部的销售数据
sales = pd.read_csv('Advertising.csv')
# 查看数据的前5行
sales.head()

# 数据集中各变量的描述性统计分析
sales.describe()
```

通过数据的描述性统计分析，我们可以得到这些数值变量的基本统计值，如均值、最小值、最大值、下四分位、上四分位、标准差，而这些统计值有助于你对数据的理解和分布的解读。接下来需要根据读取进来的数据构造回归模型，但建模之前，我们一般需要将数据集拆分成训练集（用于建模）和测试集（用于模型的评估）两个部分。

```
# 抽样--构造训练集和测试集
Train,Test = train_test_split(sales, train_size = 0.8, random_state=1234)

# 建模
fit = smf.ols('sales~TV+radio+newspaper', data = Train).fit()

# 模型概览的反馈
fit.summary()
```

通过模型反馈的结果我们可知，模型是通过显著性检验的，即F统计量所对应的P值是远远小于0.05这个阈值的，说明需要拒绝原假设（即认为模型的所有回归系数都不全为0）。

在上一期的文章中，我们说过，模型的显著性通过检验的话，并不代表每一个自变量都对因变量是重要的，所以还需要进行偏回归系数的显著性检验。通过上图的检验结果显示，除变量newspaper对应的P值超过0.05，其余变量都低于这个阈值，说明newspaper这个广告渠道并没有影响到销售量的变动，故需要将其从模型中剔除。

```
# 重新建模
fit2 = smf.ols('sales~TV+radio', data = Train.drop('newspaper', axis = 1)).fit()

# 模型信息反馈
fit2.summary()
```

通过第二次建模（模型中剔除了newspaper这个变量），结果非常明朗，一方面模型通过了显著性检验，另一方面，所有的变量也通过了显著性检验。那问题来了，难道你剔除了newspaper这个变量后，模型效果确实变好了吗？验证一个模型好不好，只需要将预测值和真实值做一个对比即可，如果模型越优秀，那预测出来的结果应该会更接近与现实数据。接下来，我们就基于fit和fit2这两个模型，分别在Test数据集上做预测：

```
# 第一个模型的预测结果
pred = fit.predict(exog = Test)

# 第二个模型的预测结果
pred2 = fit2.predict(exog = Test.drop('newspaper', axis = 1))

# 模型效果对比
RMSE = np.sqrt(mean_squared_error(Test.sales, pred))
RMSE2 = np.sqrt(mean_squared_error(Test.sales, pred2))

print('第一个模型的预测效果：RMES=%.4f\n' %RMSE)
print('第二个模型的预测效果：RMES=%.4f\n' %RMSE2)
```

对于连续变量预测效果的好坏，我们可以借助于RMSE(均方根误差，即真实值与预测值的均方根）来衡量，如果这个值越小，就说明模型越优秀，即预测出来的值会越接近于真实值。很明显，模型2的RMSE相比于模型1会小一些，模型会更符合实际。最后，我们再利用可视化的方法来刻画真实的观测点与拟合线之间的关系：

```
# 真实值与预测值的关系
# 设置绘图风格
plt.style.use('ggplot')
# 设置中文编码和负号的正常显示
plt.rcParams['font.sans-serif'] = 'Microsoft YaHei'

# 散点图
plt.scatter(Test.sales, pred, label = '观测点')
# 回归线
plt.plot([Test.sales.min(), Test.sales.max()], [pred.min(), pred.max()], 'r--', lw=2, label = '拟合线')

# 添加轴标签和标题
plt.title('真实值VS.预测值')
plt.xlabel('真实值')
plt.ylabel('预测值')

# 去除图边框的顶部刻度和右边刻度
plt.tick_params(top = 'off', right = 'off')

# 添加图例
plt.legend(loc = 'upper left')
# 图形展现
plt.show()
```

从上面的关系图来看，模型确实拟合的还是蛮不错的，这些真实点基本上都在拟合线附近，并没有产生太大的差异。

以上所分享的案例，全都是通过Python工具完成分析和建模的落地，接下来我们再利用R语言复现一遍，这里只贴上脚本，就不作详细的介绍了，如果有任何疑问都可以在公众号的后台给我留言。

利用R语言建模

```
# 读取数据
sales <- read.csv('C:\\Users\\Administrator\\Desktop\\Advertising.csv')
```

```
# 数据的描述性统计
summary(sales)


# 抽样
set.seed(1234)
index <- sample(1:nrow(sales), size = 0.8*nrow(sales))

train <- sales[index,]
test <- sales[-index,]

# 建模
fit <- lm(sales ~ ., data = sales)
# 模型概览信息
summary(fit)
```

111111111111111

```
# 模型修正
fit2 <- lm(sales ~ TV + radio, data = sales)
# 模型概览信息
summary(fit2)
```

111111111111111111

```
# 第一个模型预测
vars <- c('TV','radio','newspaper')
pred <- predict(fit, newdata = test[, vars])

# 第二个模型预测
# 模型预测
vars2 <- c('TV','radio')
pred2 <- predict(fit2, newdata = test[, vars2])

# 预测效果评估 RMSE
RMSE <- function(x,y){
  sqrt(mean((x-y)^2))
}

RMSE1 <- RMSE(test$sales, pred)
RMSE2 <- RMSE(test$sales, pred2)

RMSE1;RMSE2
```

11111111111111111111

```
# 预测值与实际值的对比
plot(test$sales,pred2, type = 'p', pch = 20, col = 'steelblue',
     xlab = '真实值', ylab = '预测值', main = '真实值VS.预测值')
lines(x = c(min(test$sales),max(test$sales)),
     y = c(min(pred2), max(pred2)),
     lty=2, col = 'red', lwd = 2)
```

111111111111111