



**题目: 汽车效能指标的线性回归分析**

**姓 名 杜泽宣**

**学 院 理学院**

**专 业 数学与应用数学**

**班 级 2021214101**

**学 号 2021212487**

**指导教师 张琳**

# **2023年 11月**

**汽车效能指标的线性回归分析**

**摘 要**

本论文研究了燃油效率关于各种预测变量的变化情况,使用了线性回归、岭回归进行预测,尝试处理了cylinder、hp、displacement之间的多重共线性问题,同时构建神经网络进行机器学习,建立Sequential模型,使用反向传播算法得到更加精准的预测结果.

**关键词** 线性回归 岭回归 多重共线性 机器学习 神经网络 反向传播算法

**Linear regression analysis of automobile efficiency indicators**

**ABSTRACT**

In this thesis, fuel efficiency was studied about the changes of various predictor variables, linear regression, ridge regression were used for prediction, tried to deal with the problem of multiple covariance between cylinder, hp, displacement, and at the same time, constructed neural network for machine learning, established Sequential model, and used the back propagation algorithm to get more accurate prediction results.

**KEY WORDS** Linear regression Ridge regression Multicollinearity Machine learning Neural networks Backpropagation algorithms

**第一章 引言**

## 1.1 背景介绍

在20世纪70年代和80年代,汽车工业经历了许多变革. 由于石油危机的影响,对燃油效率的关注逐渐增加. 政府对燃油效率标准的制定和监管也成为一个重要议题. 在这一时期,汽车制造商面临着挑战,需要设计和生产更加燃油高效的车辆以满足当时的市场需求和法规要求.

"Auto MPG"数据集记录了这个时期的许多汽车型号的信息,包括它们的MPG、发动机规格以及其他关键特征. 通过对这些数据进行线性回归分析,我们可以探索不同特征与汽车燃油效率之间的关系,了解哪些因素对燃油效率产生影响,并根据这些数据对比现在汽车设计数据,了解汽车工业中发动机设计制造的发展.

## 1.2 目标和解决方法

### 1.2.1目标

我们的目标是建立一个线性回归模型,通过分析发动机气缸数、发动机排量、马力、车辆重量、加速度等特征与汽车燃油效率之间的关系,来预测汽车的燃油效率. 这将有助于汽车制造商更好地了解如何优化设计,以提高车辆的燃油经济性,满足市场需求和法规标准.

### 1.2.2 解决方法

我们将使用线性回归模型来建立特征与MPG之间的数学关系. 通过分析回归系数,我们可以确定哪些特征对于预测燃油效率最为重要. 此外,我们还可以评估模型的准确性,并探索是否有必要考虑非线性关系或交互作用.

通过这样的分析,我们可以为汽车制造商提供有关如何改进车辆设计以提高燃油效率的洞察,同时也有助于消费者更好地理解汽车性能与燃油经济性之间的关系.

##### 1.3 线性回归方法研究现状

# **1.3.1 理论研究现状**

最小二乘法: 线性回归最早的形式是基于最小二乘法的. 这个方法最小化了观测值与模型预测值之间的残差平方和,从而找到最适合数据的直线. 这一理论基础奠定了线性回归的基础.

经典线性回归假设: 经典线性回归建立在一系列假设之上,包括线性关系、独立同分布误差、常数方差和正态分布误差等. 这些假设为线性回归的统计推断提供了理论基础.

参数估计与推断: 统计学理论关注如何对线性回归模型中的参数进行估计,并提供了推断的方法,如置信区间和假设检验. 这有助于评估模型参数的精度和可靠性.

多重共线性和过拟合: 统计学理论研究了共线性(自变量之间高度相关)和过拟合(模型过度适应训练数据)等问题,并提供了处理这些问题的方法,如岭回归和套索回归.

广义线性模型: 统计学家推广了线性回归模型,引入了广义线性模型(GLM),以处理非正态分布的响应变量,如二项分布、泊松分布等.

# **1.3.2 机器学习算法研究现状**

大数据和分布式计算: 随着大规模数据集的出现,机器学习实践中对线性回归的研究集中在如何有效处理大规模数据、利用分布式计算和流式学习等方面.

特征工程和自动化: 在机器学习实践中,特征工程是一个关键问题. 研究者们研究如何自动进行特征选择和提取,以及如何处理高维数据,这些都对线性回归模型的性能有着直接的影响.

深度学习中的线性关系: 在深度学习中,研究者们关注如何利用神经网络中的线性关系. 线性激活函数和线性层等结构被用于在更复杂的深度模型中引入线性关系.

解释性和可解释性: 在机器学习中,特别是在应用中,解释性和可解释性对于模型的成功应用至关重要. 研究者们致力于提高线性回归模型的解释性,使其更易于理解和接受.

在线学习和增量学习: 机器学习实践中对于在线学习和增量学习的研究使得模型能够在接收到新数据时不断进行学习和更新,而无需重新训练整个模型.

# **第二章 分析过程所需研究方法介绍**

##### 2.1 线性回归

# **2.1.1 最小二乘法**

1. 模型假设

我们假设一个形式为  的模型,其中:

是观测值

 是模型函数,参数为 

$epsilon$ 是误差项

2. 计算残差平方和(SSR):

对于每个数据点 ,残差为 . 目标是最小化残差平方之和:

3. 最小化SSR:

我们找到能使 SSR 最小化的  值:.

4. 线性最小二乘法:

对于线性模型 ,回归方程为

 , 

求解  和  得到最小二乘估计值.

# **2.1.2 交互变量**

交互变量反映了一个变量对结果的影响如何取决于另一个变量的值. 它们是通过将两个或多个现有变量相乘而产生的.

使用交互变量处理协变量的步骤:

1.确定潜在的协变量:

2.考虑可能混淆或影响所研究关系的变量.

3.这些变量可能同时与自变量和因变量相关系

4.创建交互项:

将感兴趣的自变量与每个已确定的协变量相乘,创建交互项.

5.在模型中加入交互项:

将交互项与自变量和协变量的主效应一起添加到回归模型中.

6.解释结果:

如果交互项显著: 自变量对因变量的影响随协变量值的变化而变化.

如果不显著: 自变量对协变量不同值的影响相对一致.

通过有效使用交互变量,您可以更细致地了解数据中变量之间的关系,同时考虑协变量的潜在影响.

# **2.1.3 岭回归**

模型设置:

多元线性回归模型:

目标:估计系数β以最小化残差平方和(SSR):

岭回归方法:

向 SSR 添加参数:

λ(正则化参数)控制收缩量:

λ 越大:收缩率越大,方差越小,偏差越大

λ 更小:收缩率更小、方差更大、偏置更小

估计系数:

使用迭代方法(例如,梯度下降):.

岭回归的优点:

有效处理多重共线性

提高模型稳定性和预测准确性

收缩率可减少过拟合

注意事项:

仔细选择λ至关重要

交叉验证通常用于选择最优λ,由于收缩而谨慎解释系数

##### 2.2 机器学习及神经网络

# **2.2.1 模型构建**

定义问题:确定您希望网络实现的目标.

选择网络架构:选择层数、每层神经元数和激活函数.  常见的架构包括多层感知器 (MLP)、卷积神经网络 (CNN) 和循环神经网络 (RNN).

实现网络:使用 TensorFlow、PyTorch 等库以您选择的编程语言实现网络逻辑(本文使用TensorFlow).  这涉及定义前向传递(信息如何流经网络)和后向传递(如何将错误传播回更新权重).

训练网络:为网络提供训练数据和优化器算法,以调整网络中的权重,直到将训练数据的误差降至最低.

评估和改进:使用看不见的数据测试网络并分析其性能.  您可以调整网络和架构的超参数,以提高准确性和泛化性.

# **2.2.2 神经网络**

1. 首先,通过使用Keras库,创建一个Sequential模型,它是一个线性堆叠的神经网络模型. 这意味着我们可以按顺序添加一层一层的神经网络层. 第一层是一个全连接层,它包含64个神经元并使用ReLU激活函数. 这一层是输入层,所以需要指定输入数据的形状,这里是根据训练数据集中的特征数量来确定的. 第二层也是一个全连接层,同样包含64个神经元,并使用ReLU激活函数. 最后一层是输出层,包含一个神经元,用于预测. 输出层不使用激活函数,因为我们要进行回归任务,直接输出预测值.

2. 选择了RMSprop优化器,用于在训练过程中更新模型的权重以最小化损失函数.

3. 损失函数选取了均方误差(MSE). 我们的目标是最小化预测值与实际值之间的平方误差.

4.计算了两个指标用于监视训练,平均绝对误差MAE和均方误差MSE,以便在训练过程中监测模型的性能.

神经网络模型包括两个隐藏层和一个输出层,通过反向传播算法不断调整权重和偏差,以使模型的预测结果尽量接近实际的汽车燃油效率. 模型的训练过程中,使用RMSprop优化器来最小化均方误差损失函数,同时监测MAE和MSE指标,以评估模型性能.

# **第三章 问题分析与解决**

##### 3.1 数据清洗

# **3.1.1** **导入包**

1. *# 导入包*
2. import pandas as pd
3. import numpy as np
4. import seaborn as sns
5. import matplotlib.pyplot as plt
6. plt.rcParams['font.sans-serif'] = 'SimHei' *## 设置中文显示*
7. from scipy.stats import f\_oneway
8. from scipy.stats import ttest\_ind

导入相关包,同时解决中文显示问题.

# **3.1.2 数据预处理**

导入数据

1. *# 导入数据*
2. df = pd.read\_csv('D:桌面\\Automobile.csv'

检查数据是否存在缺失,如存在,则剔除.

1. *#处理缺失值*
2. missing\_values = df.isnull().sum()
3. *# 过滤出有缺失值的列*
4. columns\_with\_missing\_values = missing\_values[missing\_values > 0]
5. if not columns\_with\_missing\_values.empty:
6. print("以下列存在缺失值:")
7. print(columns\_with\_missing\_values)
8. else:
9. print("文件中没有缺失值.")

输出:



此数据集不存在错误或缺失值,不需要删除数据.

# **3.1.3 数据可视化,处理异常值**

检查数据类型,调整数据类型,做箱线图.

1. num\_cols = ['mpg', '气缸数', '排量', 'Hp', '重量', '加速', '年份']
2. for col in num\_cols:
3. plt.figure(figsize=(8, 4))
4. sns.boxplot(df[col])
5. plt.title(f'{col}箱线图 ')
6. plt.show()

输出:

图形用户界面, 形状

描述已自动生成 图表, 箱线图

描述已自动生成

图表, 箱线图

描述已自动生成

可以从图中看出,’’Hp”项存在异常值,现处理Hp异常值.

首先,计算’’马力”(horsepower)的四分位距(IQR),定义正常值范围,限制异常值范围.

1. Q1\_hp = df['Hp'].quantile(0.25)
2. Q3\_hp = df['Hp'].quantile(0.75)
3. IQR\_hp = Q3\_hp Q1\_hp
4. lower\_bound\_hp = Q1\_hp 1.5 \* IQR\_hp
5. upper\_bound\_hp = Q3\_hp + 1.5 \* IQR\_hp
6. df['Hp'] = df['Hp'].clip(lower=lower\_bound\_hp, upper=upper\_bound\_hp)

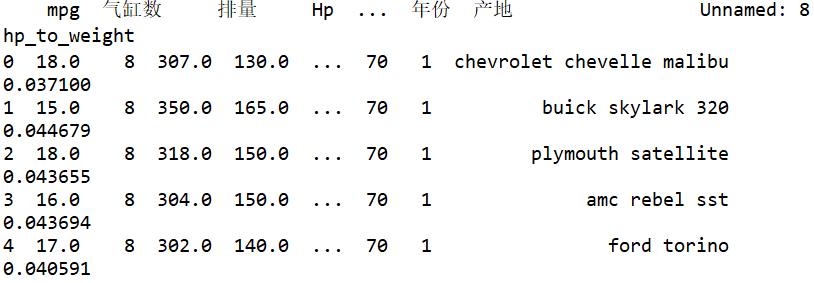
同样的,对’’重量”进行处理

1. Q1\_weight = df['重量'].quantile(0.25)
2. Q3\_weight = df['重量'].quantile(0.75)
3. IQR\_weight = Q3\_weight Q1\_weight
4. lower\_bound\_weight = Q1\_weight 1.5 \* IQR\_weight
5. upper\_bound\_weight = Q3\_weight + 1.5 \* IQR\_weight
6. df['重量']=df['重量'].clip(lower=lower\_bound\_weight,upper=upper\_bound\_weight)

根据常见的汽车数据分析,我们加入一项参数:推重比(单位Hp/kg)

1. df['hp\_to\_weight'] = df['马力'] / df['重量']
2. df.head()

输出:

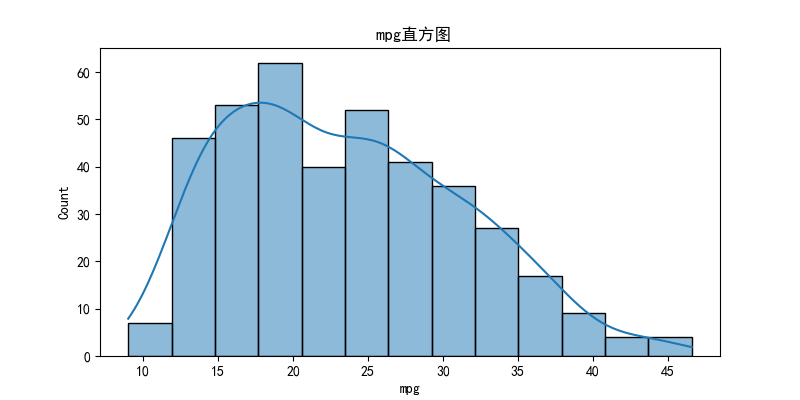


做各个变量的直方图:

代码:

1. num\_cols = ['mpg', '气缸数', '排量', '马力', '重量', '加速', '车型年份', 'hp\_to\_weight']
2. for col in num\_cols:
3. plt.figure(figsize=(8, 4))
4. sns.histplot(df[col], kde=True)
5. plt.title(f' {col}直方图')
6. plt.show()

输出:

图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

描述已自动生成图表, 直方图

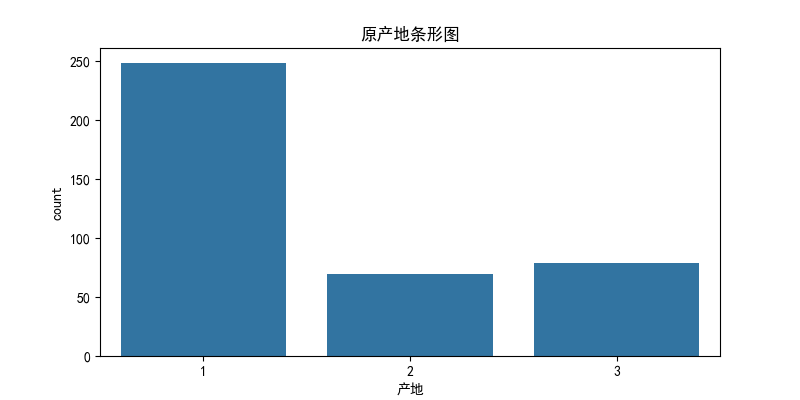
描述已自动生成图表, 直方图

描述已自动生成

生成产地条形图:

代码:

1. plt.figure(figsize=(8, 4))
2. sns.countplot(x='产地', data=df)
3. plt.title('原产地条形图')
4. plt.show()

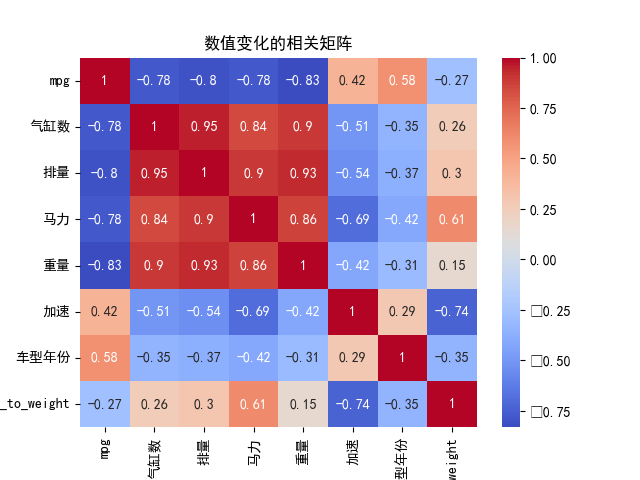


建立数值变化相关性矩阵:

代码:

1. *#计算数值变量之间的相关系数.*
2. corr\_matrix = df[num\_cols].corr()
3. *# 显示相关矩阵*
4. sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm')
5. plt.title('数值变化的相关矩阵')
6. plt.show()

输出:



##### 3.2 数据分析

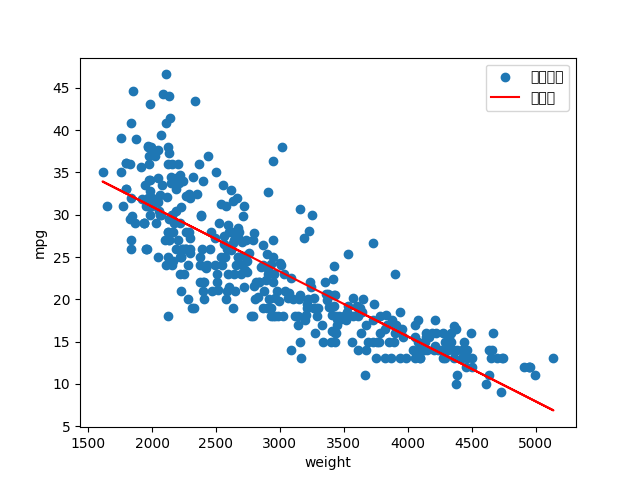
# **3.2.1 单变量回归分析**

以mpg为因变量,分别以displacement,weight,accelation为因变量,做回归分析.

1. import pandas as pd
2. from sklearn.linear\_model import LinearRegression
3. import matplotlib.pyplot as plt
4. *#读取数据集*
5. data = pd.read\_csv('F:\\Desktop\\auto-mpg.csv')
6. *#准备数据*
7. *# 分别选择cylinder, displacement, weight, acceleration作为自变量*
8. *# 每个自变量进行一次单变量回归*
9. independent\_variables = ['cylinder', 'displacement', 'weight', 'acceleration', 'hp']
10. dependent\_variable = 'mpg'
11. *#循环进行回归分析*
12. for var in independent\_variables:
13. *# 准备自变量和因变量*
14. X = data[[var]]
15. y = data[dependent\_variable]
16. *# 创建线性回归模型*
17. model = LinearRegression()
18. *# 拟合模型*
19. model.fit(X, y)
20. *# 打印回归结果*
21. print(f"因变量: {dependent\_variable}, 自变量: {var}")
22. print(f"斜率 (回归系数): {model.coef\_[0]}")
23. print(f"截距: {model.intercept\_}")
24. print(f"R-squared (拟合优度): {model.score(X, y)}")
25. print("\n")
26. *# 绘制散点图和回归线*
27. plt.scatter(X, y, label='实际数据')
28. plt.plot(X, model.predict(X), color='red', label='回归线')
29. plt.xlabel(var)
30. plt.ylabel(dependent\_variable)
31. plt.legend()
32. plt.show()

# 输出:

图表, 散点图

描述已自动生成图表, 散点图

描述已自动生成图表, 散点图

描述已自动生成

三条直线的表达式为:

mpg-displacement





mpg-weight





mpgacceleration





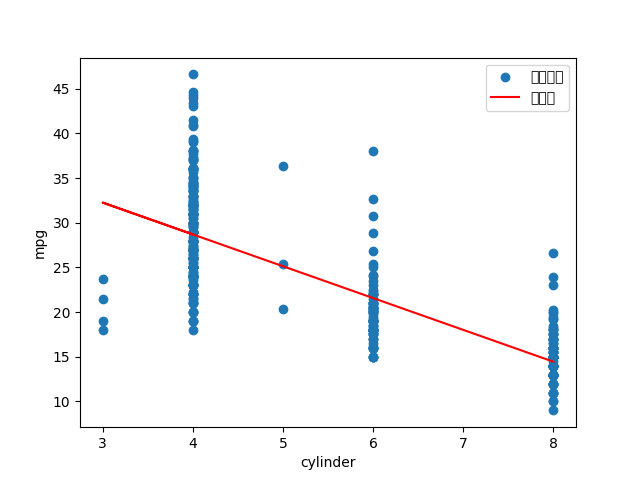
mpg-hp





可以看出,mpg关于weight,displacement拟合情况较好,而mpg-acceleration拟合度很差.

同时,根据mpg-cylinder散点图:



可以看出:

大部分车企造车都是4,6,8缸.3缸和5缸数量和仅为7,样本量太少,应当剔除.

我们再做一次回归(代码同上),新的方程如下:

mpg-displacement





mpg-weight





mpg-acceleration





mpg-hp





拟合优度值均有所提高.

# **3.2.2 多变量回归**

1. *# 选择自变量和因变量*
2. X = data[['cylinder', 'displacement', 'hp', 'weight', 'acceleration']]
3. y = data['mpg']
4. *# 创建多变量线性回归模型*
5. model = LinearRegression()
6. model.fit(X, y)
7. *# 打印回归系数和截距*
8. print("Coefficients:")
9. for feature, coef in zip(X.columns, model.coef\_):
10. print(f"{feature}: {coef}")
11. print(f"Intercept: {model.intercept\_}")

拟合结果­­\*:





可以看出,拟合优度值有一定的提高.

考虑到数据集中多种变量可能出现的共线性问题,加入交互变量进行回归.

根据客观认知,汽车缸数越高,排量应当更大,即displacement值更高,做cy-dis图观察:

cy和dis做为一个联合变量(cy·dis)加入变量进行回归





我们又想到加速能力与推重比(hp/weight)相关,我们观察一下数据变化:

线性性较强,创建(acc·hp/wei)加入变量进行回归:



\*cy=cylinder,dis=displacement,wei=weight,acc=acceleration,后面均以此代替

# **3.2.3 尝试建立岭回归**

看特征工程中的热力图,可以发现cy, wei, hp, dis之间存在较强线性关系,可以考虑使用岭回归对这部分数据进行处理:

尝试对cy,dis,hp,wei三个变量进行岭回归

1. import pandas as pd
2. import numpy as np
3. from sklearn.linear\_model import Ridge
4. from sklearn.preprocessing import StandardScaler
5. from sklearn.metrics import mean\_squared\_error
6. from sklearn.model\_selection import GridSearchCV
7. *# 创建一个示例数据集*
8. data = pd.read\_csv('F:\\Desktop\\test\\auto-mpg.csv')
9. df = pd.DataFrame(data)
10. *# 定义预测变量和响应变量*
11. X = df[["cylinder", "displacement", "hp", "weight"]]
12. y = df["mpg"]
13. *# 数据预处理:标准化特征*
14. scaler = StandardScaler()
15. X\_scaled = scaler.fit\_transform(X)
16. *# 岭回归模型*
17. ridge = Ridge()
18. *# 使用网格搜索进行交叉验证以找到最佳的岭参数*
19. parameters = {'alpha': np.logspace(-3, 3, 13)}
20. grid = GridSearchCV(ridge, parameters, cv=10)
21. grid.fit(X, y)
22. *# 最佳参数和对应的模型*
23. best\_alpha = grid.best\_params\_['alpha']
24. best\_model = grid.best\_estimator\_
25. *# 模型评估*
26. y\_pred = best\_model.predict(X)
27. mse = mean\_squared\_error(y, y\_pred)
28. print('mse:',mse)
29. print('beat\_alpha:', best\_alpha)
30. *# 模型系数*
31. coefficients = best\_model.coef\_
32. *# 截距*
33. intercept = best\_model.intercept\_
34. *# 打印岭回归方程*
35. equation = "y = {:.2f}".format(intercept)
36. for i, coef in enumerate(coefficients):
37. equation += " + {:.2f}\*x{}".format(coef, i+1)
38. *# 计算R^2值*
39. r2\_score = best\_model.score(X, y)
40. print('方程:')
41. print(equation)
42. print('R^2:', r2\_score)

得到:

mse=16.998

beat\_alpha=10.03

方程:y = 45.85 + -0.51\*cy + -0.00\*dis + -0.03\*hp + -0.01\*wei

R^2=0.7225

##### 3.3 机器学习模拟­\*

# **3.3.1 数据处理**

为每一列数据打标签,检查缺失数据,划分数据集并将Origin变量使用独立热编码,独立划分为0-1变量.再将数据标准化.

# **3.3.2 模型搭建**

模型采用线性回归模型,它包含两个紧密的隐藏层,以及返回单个、连续值的输出层. 执行模型采用网络层结构创建,以及模型编译.

第一层Dense的input\_shape为输入层大小,前边的64为该全连接层的输出层(隐藏层)——只有最后的全连接层 layers.Dense(1)输出才是输出层,否则为隐藏层.

activation为激活函数,这里使用线性激活.

关于build\_model()函数:

首先,通过keras.Sequential创建了一个Sequential模型,它是一个线性堆叠的神经网络模型. 这个模型包含了三层:

第一层是一个具有64个神经元的全连接层(Dense层),激活函数为ReLU(Rectified Linear Unit),并且指定了输入数据的形状,这里的input\_shape是train\_dataset中特征的数量.

第二层也是一个具有64个神经元的全连接层,激活函数同样为ReLU.

第三层是一个具有1个神经元的全连接层,这通常用于回归问题,输出一个单一的连续数值.

接下来,通过tf.keras.optimizers.RMSprop创建了一个RMSprop优化器.

使用model.compile来编译模型,指定了损失函数为均方误差(MSE,Mean Squared Error),优化器为上面创建的RMSprop优化器,并且定义了模型的评估指标,包括平均绝对误差(MAE,Mean Absolute Error)和均方误差(MSE).

# 

# **3.3.3 模型检测**

添加一个输出点的函数——目的是显示训练的进程(提示:本次训练拟合次数为1000.

关于于fit函数的参数:

normed\_train\_data, train\_labels:

训练所需的归一化数据,训练数据对应的标签；

epochs=EPOCHS: 拟合次数

validation\_split:用于在没有提供验正集的时候,按一定比例从训练集中取出一部分作为验证集

verbose:日志显示配置:

verbose = 0 为不在标准输出流输出日志信息

verbose = 1 为输出进度条记录

verbose = 2 为每个epoch输出一行记录

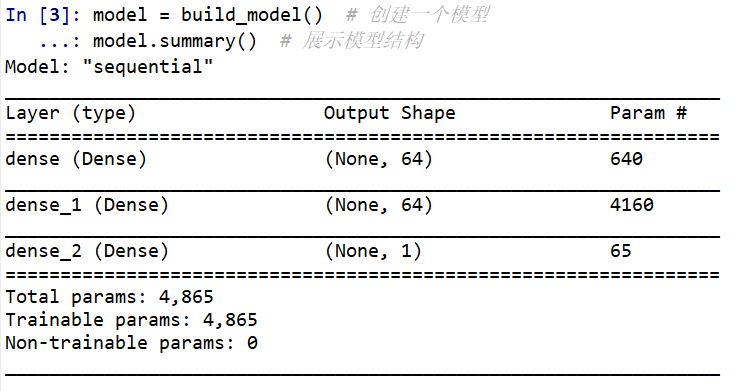
callbacks:训练中所做的操作

我们可以从example\_result中看出拟合值与真实值之间的差比较小,可以认为使用该模型的拟合较为成功.

# **3.3.4 代码**

1. import pathlib
2. import numpy as np *#numpy==1.18*
3. import matplotlib.pyplot as plt
4. import pandas as pd
5. import seaborn as sns
6. import tensorflow as tf *#tensorflow==2.1.0*
7. from tensorflow import keras
8. from tensorflow.keras import layers
9. dataset\_path = pd.read\_csv('F:\\Desktop\\test\\auto-mpg.csv')
10. *#数据处理*
11. column\_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
12. 'Acceleration', 'Model Year', 'Origin']
13. raw\_dataset = pd.read\_csv(dataset\_path, names=column\_names, na\_values = "?", comment='\t', sep=" ", skipinitialspace=True)
14. dataset = raw\_dataset.copy()  *# 复制读取的数据,避免修改元数据*
15. *# dataset.head()*
16. *# dataset.isna().sum()  # 显示无效数据分布情况*
17. dataset = dataset.dropna()  *# 移除无效数据*
19. *# dataset.isna().sum()  # 显示无效数据分布情况*
20. origin = dataset.pop('Origin')  *# 获取分类序号——以供one-hot转换需要*
21. *# origin*
22. dataset['USA'] = (origin == 1)\*1.0
23. *# one-hot转换:只有当列数据中值为1时,才设为1,否则为0,也就相当于一整列数据按 (origin == 1)条件转换为1 or 0*
24. dataset['Europe'] = (origin == 2)\*1.0
25. dataset['Japan'] = (origin == 3)\*1.0
26. *# dataset.head()*
27. *#训练集和测试集*
28. train\_dataset = dataset.sample(frac=0.8,random\_state=0)  *# 拆分获得训练数据*
29. test\_dataset = dataset.drop(train\_dataset.index)  *# 拆分获得测试数据*
30. train\_stats = train\_dataset.describe()  *# 获取数据的一系列描述信息*
31. train\_stats.pop("MPG")  *# 移除MPG的数据列*
32. train\_stats = train\_stats.transpose()  *# 行列转换——也就是相当于翻转*
33. *#分离训练数据的标签*
34. train\_labels = train\_dataset.pop('MPG')   *# 将移除的MPG列数据返回,赋值给train\_labels*
35. test\_labels = test\_dataset.pop('MPG')
36. train\_labels   *# 显示数据*
37. *# train\_labels*
38. *# test\_labels*
39. *#数据标准化*
40. def norm(x):
41. return (x train\_stats['mean']) / train\_stats['std']
42. normed\_train\_data = norm(train\_dataset)  *# 获取训练数据中的归一化数据*
43. normed\_test\_data = norm(test\_dataset)  *# 获取测试归一化数据*
44. *#模型构建*
45. *# 第一层Dense的input\_shape为输入层大小,前边的64为该全连接层的输出层(隐藏层)——只有最后的全连接层 layers.Dense(1)输出才是输出层,否则为隐藏层.*
46. *# activation为激活函数——这里是线性激活*
47. def build\_model():
48. model = keras.Sequential([
49. layers.Dense(64, activation='relu', input\_shape=[len(train\_dataset.keys())]),
50. layers.Dense(64, activation='relu'),
51. layers.Dense(1)
52. ])
53. optimizer = tf.keras.optimizers.RMSprop(0.001)
54. model.compile(loss='mse',
55. optimizer=optimizer,
56. metrics=['mae', 'mse'])
57. return model
58. *#模型检测*
59. example\_batch = normed\_train\_data[:10]  *# 获取十个数据来预测*
60. example\_result = model.predict(example\_batch)
61. *#example\_result*
62. *# 通过为每个完成的时期打印一个点来显示训练进度*
63. class PrintDot(keras.callbacks.Callback):
64. def on\_epoch\_end(self, epoch, logs):
65. if epoch % 100 == 0: print('')
66. print('.', end='')
68. EPOCHS = 1000  *# 拟合次数*
69. *# 返回的history为一个对象,内包含loss信息等*
70. history = model.fit(
71. normed\_train\_data, train\_labels,
72. epochs=EPOCHS, validation\_split = 0.2, verbose=0,
73. callbacks=[PrintDot()])
74. hist = pd.DataFrame(history.history)  *# 返回一个DataFrame对象,包含history的history数据*
75. hist['epoch'] = history.epoch  *# 末尾添加一列数据包含epoch信息*
76. *# hist.tail()  # 展示表单数据*
77. *# 训练过程可视化*
78. def plot\_history(history):
79. hist = pd.DataFrame(history.history)
80. hist['epoch'] = history.epoch
81. plt.figure()  *# 创建新窗口*
82. plt.xlabel('Epoch')  *# x轴标签(名字)*
83. plt.ylabel('Mean Abs Error [MPG]')  *# y轴标签(名字)*
84. plt.plot(hist['epoch'], hist['mae'],
85. label='Train Error')  *# 绘图,label为图例*
86. plt.plot(hist['epoch'], hist['val\_mae'],
87. label = 'Val Error')  *# 绘图,label为图例*
88. plt.ylim([0,5])  *# y轴长度限制*
89. plt.legend()  *# 展示图例*
90. plt.figure()
91. plt.xlabel('Epoch')
92. plt.ylabel('Mean Square Error [$MPG^2$]')
93. plt.plot(hist['epoch'], hist['mse'],
94. label='Train Error')
95. plt.plot(hist['epoch'], hist['val\_mse'],
96. label = 'Val Error')
97. plt.ylim([0,20])
98. plt.legend()
99. plt.show()  *# 必须要有show才能显示绘图*
100. *# plot\_history(history)*
101. *# 模型改进*
103. model = build\_model()  *# 重新定义模型*
104. *# patience 值用来检查改进 epochs 的数量*
105. *# 定义callbacks的操作设置,这里采用了每10次fit,进行一次判断是否停下,判断依据是当val\_loss不改变甚至降低.*
106. early\_stop = keras.callbacks.EarlyStopping(monitor='val\_loss', patience=10)
107. history = model.fit(normed\_train\_data, train\_labels, epochs=EPOCHS,
108. validation\_split = 0.2, verbose=0, callbacks=[early\_stop, PrintDot()])
109. loss, mae, mse = model.evaluate(normed\_test\_data, test\_labels, verbose=2)  *# 返回第一个数据loss为损失值*
110. print("Testing set Mean Abs Error: {:5.2f} MPG".format(mae))  *# mae为我们构建网络归回预测的值——即预测的MPG*
111. *# plot\_history(history)*
112. *# 模型评估*
113. loss, mae, mse = model.evaluate(normed\_test\_data, test\_labels, verbose=2)  *# 返回第一个数据loss为损失值*
114. print("Testing set Mean Abs Error: {:5.2f} MPG".format(mae))  *# mae为我们构建网络归回预测的值——即预测的MPG*
115. *# 训练数据可视化*
116. test\_predictions = model.predict(normed\_test\_data).flatten()
117. *# 预测信息会被flatten()平铺展开后以一维数据返回*
118. plt.scatter(test\_labels, test\_predictions)  *# 绘制散点图*
119. plt.xlabel('True Values [MPG]')
120. plt.ylabel('Predictions [MPG]')
121. plt.axis('equal')
122. plt.axis('square')
123. plt.xlim([0,plt.xlim()[1]])
124. plt.ylim([0,plt.ylim()[1]])
125. \_ = plt.plot([-100, 100], [-100, 100])  *# 画一条y = x的直线,方便分析*
126. *#误差分布*
127. error = test\_predictions test\_labels  *# test\_labels 是原始的MPG值序列,所以相减得到误差.*
128. plt.hist(error, bins = 25)  *# 画矩形图——bins表示每次绘图的最大矩形数目*
129. plt.xlabel("Prediction Error [MPG]")
130. \_ = plt.ylabel("Count")

模型运行结果:

文本

描述已自动生成文本

描述已自动生成

图形用户界面, 应用程序

描述已自动生成图形用户界面, 图表, 直方图

描述已自动生成

可以看到，模型有效准确值稳定在了一定范围后，训练就停止了。

图片包含 文本

描述已自动生成

说明MPG稳定在1.84左右

图表, 散点图

描述已自动生成

MPG数据基本集中于该直线

图表, 直方图

描述已自动生成

误差主要分布在-2~2之间，0最多。说明训练模型较为合适。

注:所有被’#’注释的单行代码需要在终端中运行

# **第四章 总结**

##### 4.1 总结与分析

随着时间的推移，燃油效率:平均每加仑英里数(mpg)似乎随着时间的推移而增加，这表明汽车变得更加省油。这可能是由于技术的进步和汽车制造业对燃油效率的日益关注。

马力和重量:马力和重量之间似乎存在正相关关系，表明较重的汽车往往拥有更强劲的发动机。然而，马力和重量似乎都与mpg负相关，这表明较重的汽车和发动机功率更大的汽车往往更省油。

产地和燃油效率:我们的假设检验表明，不同产地的汽车平均每加仑汽油行驶里程有显著差异。这表明汽车的生产地区可能会对其燃油效率产生影响。

参考实际生活，加入生活常见的推重比，可能会为这些变量和mpg之间的关系提供不同的预测结果。

当数字输入数据特征的值存在不同范围时，每个特征应独立缩放到相同范围。（归一化，一般采用独立热编码形式）