Noah Nicol

CU Boulder

November 2023

Introduction to Machine Learning: Supervised Learning Assignments

Week 2: Multiple Linear Regression

%matplotlib inline

import numpy as np

import scipy as sp

import scipy.stats as stats

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Set color map to have light blue background

sns.set()

import statsmodels.formula.api as smf

import statsmodels.api as sm

columns = ['mpg','cylinders','displacement','horsepower','weight','acceleration','model\_year','origin','car\_name']

df = pd.read\_csv("data/auto-mpg.data", header=None, delimiter=r"\s+", names=columns)

print(df.info())

df.describe()

# replace data frame with cleaned data frame

# fix data types, remove null or undefined values, drop the column car\_name

#convert 'horsepower to numeric, coerce replaces non-numeric with NaN'

df['horsepower'] = pd.to\_numeric(df['horsepower'], errors = 'coerce')

#drop rows with missing values

df.dropna()

#drop car\_name column

df\_cleaned = df.drop(columns = ['car\_name'])

df\_cleaned.head()

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import r2\_score

from sklearn.impute import SimpleImputer

# all columns except mpg (independent variables)

features = df\_cleaned.columns.difference(['mpg'])

best\_feature = None

max\_r\_squared = -1

for feature in features:

#shape features to 2D array for scikit-learn

X = df\_cleaned[feature].values.reshape(-1,1)

imputer = SimpleImputer(strategy = 'mean')

X = imputer.fit\_transform(X)

model = LinearRegression()

#fit model

model.fit(X, df\_cleaned['mpg'])

predictions = model.predict(X)

#calculate R^2

current\_r\_squared = r2\_score(df\_cleaned['mpg'], predictions)

print(feature)

print(current\_r\_squared)

#update best feature

if current\_r\_squared > max\_r\_squared:

best\_feature = feature

max\_r\_squared = current\_r\_squared

print(f"final best feature: {best\_feature}")

print(f"final max r^2: {max\_r\_squared}")

best\_predictor='weight'

best\_r\_squared= 0.6917929800341573

# return updated best\_degree and best\_r\_squared

# your code here

X = df\_cleaned['weight'].values.reshape(-1,1)

y = df\_cleaned['mpg']

best\_degree = 1

best\_r\_squared = 0

for degree in range(1,11):

#generate polynomial features

X\_poly = np.column\_stack([X\*\*i for i in range(1, degree + 1)])

#add constant term (intercept)

X\_poly = sm.add\_constant(X\_poly)

model = sm.OLS(y, X\_poly).fit()

predictions = model.predict(X\_poly)

current\_r\_squared = r2\_score(y, predictions)

#update best degree

if current\_r\_squared > best\_r\_squared:

best\_degree = degree

best\_r\_squared = current\_r\_squared

print(f"final best degree: {best\_degree}")

print(f"best R squared: {best\_r\_squared}")

best\_degree = 20

best\_r\_squared = 0.721510843572874

sound\_degree = 2

df\_cleaned['weight\_norm'] = df\_cleaned['weight']/df\_cleaned['weight'].mean()

# your code here

# return updated best\_degree and best\_r\_squared

# your code here

X = df\_cleaned['weight\_norm'].values.reshape(-1,1)

y = df\_cleaned['mpg']

best\_degree = 1

best\_r\_squared = 0

for degree in range(1,21):

#generate polynomial features

X\_poly = np.column\_stack([X\*\*i for i in range(1, degree + 1)])

#add constant term (intercept)

X\_poly = sm.add\_constant(X\_poly)

model = sm.OLS(y, X\_poly).fit()

predictions = model.predict(X\_poly)

current\_r\_squared = r2\_score(y, predictions)

print(f"Degree: {degree}")

print(f"Current R^2: {current\_r\_squared}")

#update best degree

if current\_r\_squared > best\_r\_squared:

best\_degree = degree

best\_r\_squared = current\_r\_squared

print(f"final best degree: {best\_degree}")

print(f"best R squared: {best\_r\_squared}")

votes = pd.read\_csv('data/fl2000.txt', delim\_whitespace=True, comment='#')

votes = votes[['county', 'Bush', 'Gore', 'Nader', 'Buchanan']]

votes.describe(include='all')

# plot a pair plot of the data using the seaborn library

# possible way to save the image

# plt.savefig('pair\_plot.png', dpi = 300, bbox\_inches = 'tight')

# your code here

# print(votes.describe(include = 'all'))

sns.pairplot(votes[['county', 'Bush', 'Gore', 'Nader', 'Buchanan']])

plt.show()

# uncomment and construct a multi-linear model

indep\_vars = ['Gore', 'Nader', 'Buchanan']

#constant term for intercept

X = sm.add\_constant(votes[indep\_vars])

#dependent variable

y = votes['Bush']

model = sm.OLS(y, X).fit()

print(model.summary())

# uncomment and construct multi-linear model

# your code here

indep\_vars = ['Gore', 'Nader']

#constant term for intercept

X = sm.add\_constant(votes[indep\_vars])

#dependent variable

y = votes['Bush']

model\_multi = sm.OLS(y, X).fit()

print(model\_multi.summary())

# plot the leverage vs. the square of the residual

# your code here

leverage = model\_multi.get\_influence().hat\_matrix\_diag

residuals\_squared = model\_multi.resid\*\*2

#plot it

plt.scatter(leverage, residuals\_squared, alpha = 0.5)

plt.title('Levereage vs. Square of Residual')

plt.xlabel('Leverage')

plt.ylabel('Square of Residual')

plt.show()

# uncomment and fill unusual with list of indices for high-leverage and/or high-residual points

# unusual = []

# your code here

leverage\_threshold = 0.1

residual\_squared\_threshold = 4e9

# find indices for unusual rows

high\_leverage\_idx = [i for i, lev in enumerate(leverage) if lev>leverage\_threshold]

high\_residual\_idx = [i for i, res\_sq in enumerate(residuals\_squared) if res\_sq > residual\_squared\_threshold]

unusual = list(set(high\_leverage\_idx + high\_residual\_idx))

print(unusual)

# develop your model\_final here

# model\_final =

# your code here

votes\_filtered = votes.drop(index=unusual)

X\_filtered = sm.add\_constant(votes\_filtered[indep\_vars])

y\_filtered = votes\_filtered['Bush']

model\_final = sm.OLS(y\_filtered, X\_filtered).fit()

print(model\_final.summary())

fat = pd.read\_csv('data/bodyfat.csv')

fat = fat.drop('Unnamed: 0', axis=1)

fat.Weight = fat.Weight \* 0.453592 # Convert to Kg

fat.Height = fat.Height \* 0.0254 # convert inches to m

fat['BMI'] = fat.Weight / (fat.Height\*\*2)

fat.BMI.plot.kde();

# form new table cfat and model bmi

# cfat =

# bmi =

# new table of BMI's less than 40

cfat = fat[fat['BMI'] <= 40]

#Regression variables

X = sm.add\_constant(cfat['BMI'])

y = cfat['Density']

bmi = sm.OLS(y,X).fit()

print(bmi.summary())

# plot regression model against BMI measurement

# properly label the scatterplot axs and show the regression line

# your code here

plt.scatter(cfat['BMI'], cfat['Density'], label = 'Data')

plt.plot(cfat['BMI'], bmi.predict(X), color='red', label='Regression Line')

plt.title('BMI~Density Regression Model')

plt.xlabel('BMI')

plt.ylabel('Density')

plt.legend()

plt.show()

allowed\_factors = ['Age', 'Weight', 'Height', 'Neck', 'Chest',

'Abdomen', 'Hip', 'Thigh', 'Knee', 'Ankle', 'Biceps', 'Forearm',

'Wrist']

# construct train\_fat and test\_fat from cfat dataset

# your code here

from sklearn.model\_selection import train\_test\_split

train\_fat, test\_fat = train\_test\_split(cfat, test\_size=0.8, random\_state=0)

best = ['',0]

for p in allowed\_factors:

model = smf.ols(formula='Density~'+p, data=train\_fat).fit()

print(p, model.rsquared)

if model.rsquared>best[1]:

best = [p, model.rsquared]

print('best:',best)

# uncomment and update your solution

factor = 'Abdomen'

X\_train = sm.add\_constant(train\_fat[factor])

y\_train = train\_fat['Density']

train\_bmi1 = sm.OLS(y\_train, X\_train).fit()

print(f"Adjusted R^2: {train\_bmi1.rsquared\_adj}")

# your code here

factors = ['Abdomen', 'Chest']

X2\_train = sm.add\_constant(train\_fat[factors])

y\_train = train\_fat['Density']

train\_bmi2 = sm.OLS(y\_train, X2\_train).fit()

print(f"Adjusted R^2: {train\_bmi2.rsquared\_adj}")

# your code here

factors = ['Abdomen', 'Chest', 'Weight']

X3\_train = sm.add\_constant(train\_fat[factors])

y3\_train = train\_fat['Density']

train\_bmi3 = sm.OLS(y3\_train, X3\_train).fit()

print(f"Adjusted R^2: {train\_bmi3.rsquared\_adj}")

# your code here

factors = ['Abdomen', 'Chest', 'Hip', 'Weight']

X4\_train = sm.add\_constant(train\_fat[factors])

y4\_train = train\_fat['Density']

train\_bmi4 = sm.OLS(y4\_train, X4\_train).fit()

print(f"Adjusted R^2: {train\_bmi4.rsquared\_adj}")

# your code here

# your code here

factors = ['Abdomen', 'Chest', 'Hip', 'Weight', 'Neck']

X5\_train = sm.add\_constant(train\_fat[factors])

y5\_train = train\_fat['Density']

train\_bmi5 = sm.OLS(y5\_train, X5\_train).fit()

print(f"Adjusted train R^2: {train\_bmi5.rsquared\_adj}")

X5\_test = sm.add\_constant(test\_fat[factors])

y5\_test = test\_fat['Density']

y5\_predicted = train\_bmi5.predict(X5\_test)

#calculate residual sum of squares for test data

rss\_test = ((y5\_test - y5\_predicted)\*\*2).sum()

tss\_test = ((y5\_test - y5\_test.mean()) \*\*2).sum()

n\_test = len(y5\_test)

k\_test = len(factors) +1 #+1 for constant term

adj\_r2\_test = 1- (rss\_test / (n\_test - k\_test - 1)) / (tss\_test / (n\_test -1))

print(f'Adjusted R^2 for test data: {adj\_r2\_test}')

# plot resulting adjusted rsquared vs number of predictors (k=1,2,3,4,5)

# overlay the adjusted rsquared for the test data

adjr2\_train = [0.6596864493008139, 0.6684963022754815, 0.720558794756488, 0.7359303359814342, 0.7304331649336688]

predictors = ['Abdomen', 'Chest', 'Weight', 'Hip', 'Neck']

adjr2\_test = 0.6612975258441165

plt.plot(range(1,6), adjr2\_train, marker='o', label='Training Data')

plt.scatter(6, adjr2\_test, color='red', marker='o', label='Test Data')

plt.title('Adjusted R^2 vs. Number of Predictors')

plt.xlabel('Number of Predictors (k)')

plt.ylabel('Adjusted R^2')

plt.xticks(range(1, 7), labels=predictors + ['Test Data'])

plt.legend()

plt.show()