Data Analysis and Machine-Learning

Chapter 5.1:

Feature Selection – Multicollinearity



1. Introduction

In chapter 2.1., we have seen the risk of having overfits in modelling. Although there is yet no definite, fixed methodologies to completely avoid overfitting problems, we will explore some effective ways to reduce them. Inter alia, this chapter will demonstrate methodologies associated with variable (features) controls using stochastic optimization such as multicollinearity resolutions, forward selections, backward eliminations, scaling, regularizations (L1, L2), i.e., feature selection algorithms. Let us start by importing sample datasets for step-by-step illustrations of these concepts.

2. Load Sample Dataset

#Import basic libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

#Downloading and importing sample dataset

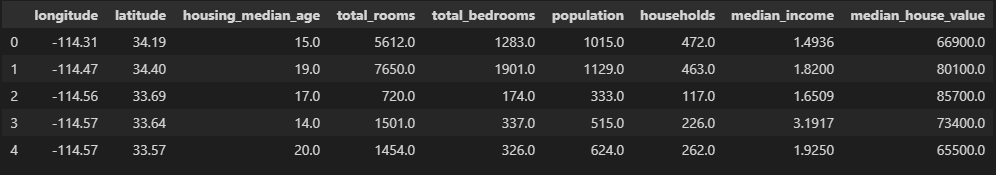
From Google Colab, you can download train and test CSV files of California housing datasets. Download and import them on your IDE.

raw\_train = pd.read\_csv('your file path\\california\_housing\_train.csv')

raw\_test = pd.read\_csv('your file path\\california\_housing\_test.csv')

#Checking the structures of the datasets

raw\_train.head()



raw\_train.columns

output:

Index(['longitude', 'latitude', 'housing\_median\_age', 'total\_rooms', 'total\_bedrooms', 'population', 'households', 'median\_income', 'median\_house\_value'], dtype='object')

Median house value is the target (Y), and the rest variables are features (variables, X) to predict Y.

#Set features and target accordingly

features = raw\_train.drop(columns='median\_house\_value')

target = raw\_train['median\_house\_value']

#Check raw model score for comparison

from sklearn.linear\_model import LinearRegression

base\_model = LinearRegression().fit(features, target)

base\_model.score(features, target)

output:

0.6413378529502691

#R2 vs adj.R2

As we have covered in chapter two, the LinearRegression model in sklearn calculates the score via R2 formula. Since there are multiple variables for this dataset (multiple regression), it is more appropriate to score via adjusted R2 method. As sklearn does not provide adj.R2 scoring algorithm, let us create the algorithm by defining function directly, as follows.

def rSquare( x, y, yhat ):

    if x.ndim == 1: p, n = 1, x.shape[0]

    else: p, n = x.shape[1], x.shape[0]

    r2 = 1 - np.sum( (y - yhat) \*\* 2) / np.sum( (y - np.mean(y)) \*\* 2 )

    adj\_r2 = 1 - (1 - r2) \* ( n - 1) / ( n - p - 1 )

    return {'r2': r2, 'adjr2': adj\_r2}

#Using created function for calculating adj.R2 score of the base model

yhat = base\_model.predict(features)

rSquare(features, target, yhat)

output:

{'r2': 0.6413378529502691, 'adjr2': 0.641168981360816}

We will compare this base model score to the other models created via various feature selection methodologies.

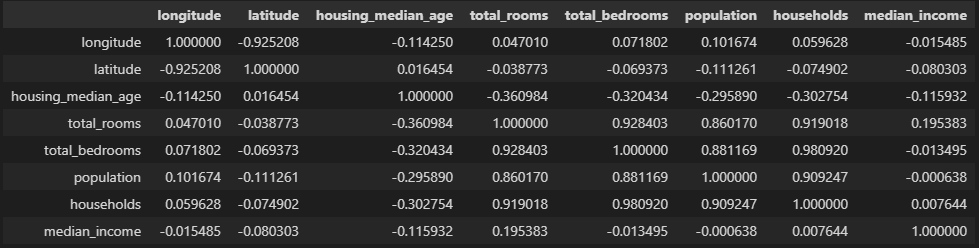
3. Multicollinearity

We have learned the concepts of correlations (correlation coefficients) and causality (regression) in chapter 2. High rates of correlations and causality among independent variables can distort the result of the model (after all, independent variables should be “independent”), which is why it is first and foremost essential to examine the rates of correlations and causality to discard highly correlated variables.

3.1. Correlation

# Correlation Coefficients

features.corr()

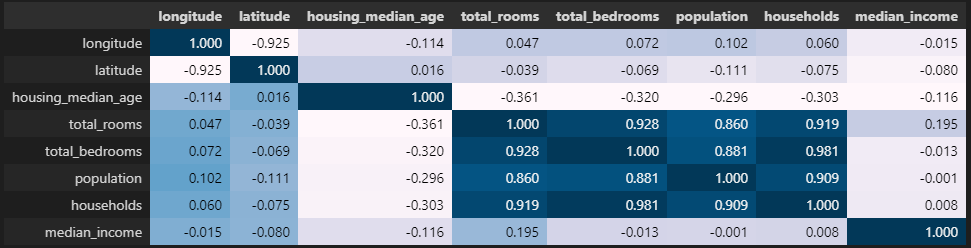


As above, Python provides simple matrix to check correlations. However, the visualization is not evident enough to grasp the whole picture in one sight.

# Customizing Correlation Matrix for better visualization (1)

You can customize the background gradients and decimal points, as follows, for better visualization;

features.corr().style.background\_gradient().set\_precision(3)



# Customizing Correlation Matrix for better visualization (2)

For even better visualization, it might be better to directly create a function for correlation matrix, as follows.

#Define function for correlation matrix

def visualCorr(x, width=15, height=7):

    plt.figure(figsize=(width,height))

    mask = np.zeros\_like(x.corr(), dtype=np.bool)

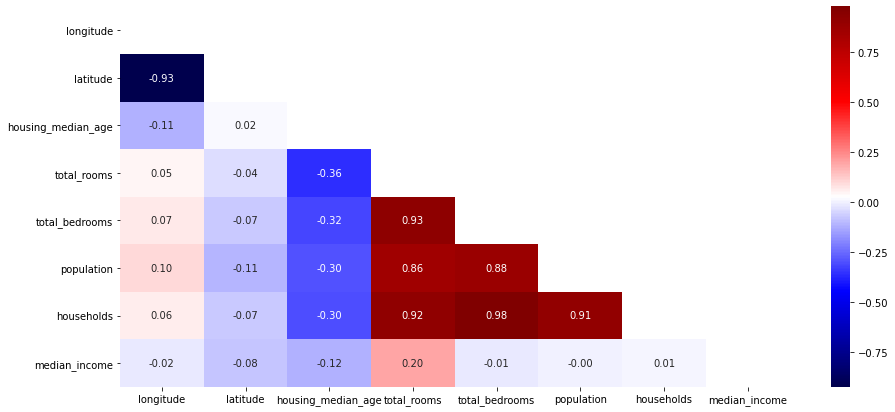
    mask[np.triu\_indices\_from(mask)] = True

    sns.heatmap(x.corr(), annot=True, fmt='.2f', mask=mask, cmap='seismic')

    plt.show()

#Setting as seismic color can be useful as it generates high contrasts between low values and high values, but you can also customize with other colors.

visualCorr(features)

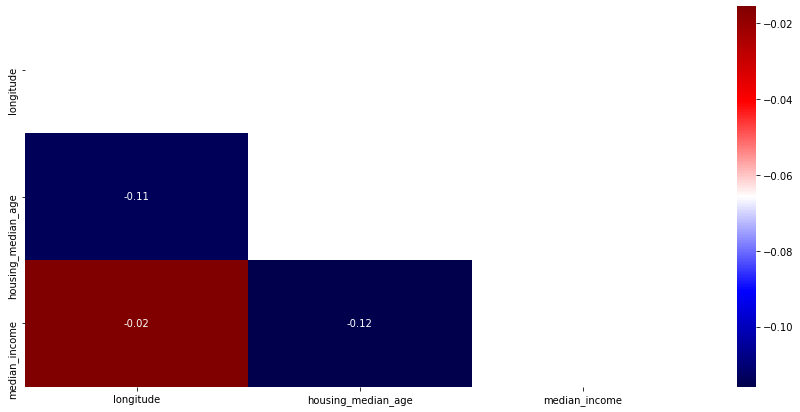


#Drop highly correlated variables

Ultimately, it is the analyst’s call to decide which variables to discard by examining correlations among variables. There is no definite threshold that becomes the standard to define “high” rates of correlations. However, it is worth remembering that simplest model with high explanation power is usually the most intuitive and convincing of all.

features\_corr = features.drop(columns=['latitude','total\_rooms','total\_bedrooms','population','population','households'])

visualCorr(features\_corr)



Now, let us compare this model with the base model.

model\_corr = LinearRegression().fit(features\_corr, target)

yhat = model\_corr.predict(features\_corr)

rSquare(features\_corr, target, yhat)

output:

{'r2': 0.5142707802464033, 'adjr2': 0.5141850431518363}

As it can be implied from the score, it is already visible that the model with these three variables is almost sufficient enough to explain quite a lot, without using many other variables (new model 0.51 vs. base model 0.64)

3.2. Causality (Variance Inflation Factor)

We have learned the concept of causality using regression models in chapter 2. VIF methodology calculates the ratio of variance of the overall model (multiple regression) to the model with single variable (simple regression). High scores of VIF imply high rates of causality among variables. Let us create a customized function in a dataframe form using Statsmodels.

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

def vif(x):

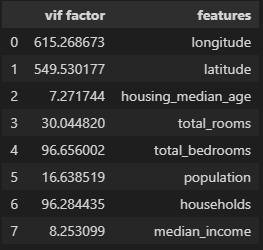
    vifFrame = pd.DataFrame()

    vifFrame['vif factor'] = [variance\_inflation\_factor(x.values, i) for i in range(x.shape[1]) ]

    vifFrame['features'] = x.columns

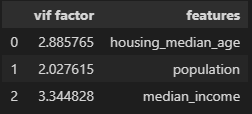
    return vifFrame

vif(features)



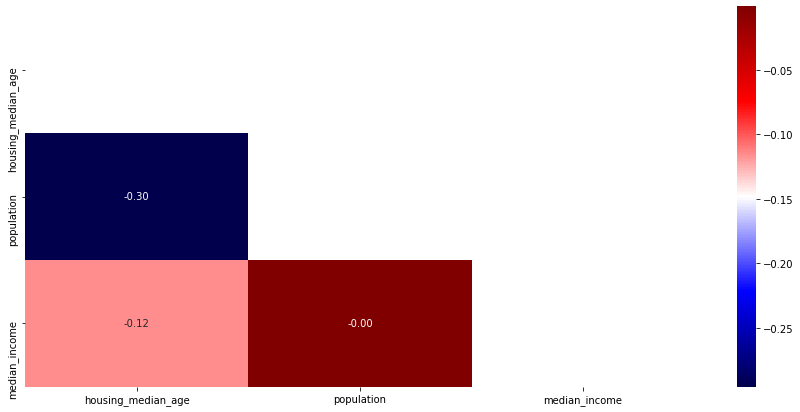
#Feature selection using VIF

vif(features.drop(columns=['longitude','latitude','households','total\_bedrooms','total\_rooms']))



#Check Correlations for VIF features

visualCorr(features\_vif)



#VIF model score

model\_vif = LinearRegression().fit(features\_vif, target)

yhat = model\_vif.predict(features\_vif)

rSquare(features\_vif, target, yhat)

output:

{'r2': 0.5150206852133975, 'adjr2': 0.5149350804861463}

As implied from the VIF model score (and similarly to the case for the Correlation model created earlier), it is clearly visible that the model with these few variables is almost sufficient enough to explain a large part of the target variable, without using various other variables [(VIF model: 0.51) vs. (base model: 0.64)].

Now that we have covered the concepts of multicollinearity, we will explore more in depth regarding feature selections in the following chapters.