Data Analysis and Machine-Learning

Chapter 5.4:

Feature Selection (4)

Scaling and Stochastic Modelling



1. Recursive Feature Elimination

I have illustrated the concepts and algorithms for forward selection, backward elimination, and stepwise selection in chapter 4.2. Recursive Feature Elimination in sklearn provides identical algorithm to the backward elimination algorithm that we created earlier. The benefits of recursive feature elimination algorithm in sklearn is that it is possible to customize the number of features to select, accordingly to the weights of the variables.

from sklearn.feature\_selection import RFE

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

feature\_selector = RFE(base\_model, n\_features\_to\_select=3)

feature\_selector = feature\_selector.fit(features, target)

display(feature\_selector.n\_features\_)

display(feature\_selector.support\_)

output:

3

array([ True, True, False, False, False, False, False, True])

It is possible to utilize the boolean values to select features using loc() function:

features.loc[:, feature\_selector.support\_]



2. Scaling

The result of the model can be distorted if the unit of dataset values are significantly different. For instance, the variables that are composed of small unit values can be miscalculated to be of low significance or low coefficients (or weights) even though they are actually more significant than other variables with larger unit values. This is why it is always significant to scale all variables in unified categories of unit values throughout the scaling process in prior to the modelling procedures.

from sklearn.preprocessing import StandardScaler

col = features.columns

scaler = StandardScaler()

features\_scaled = features.copy()

features\_scaled[col] = scaler.fit\_transform(features[col])

features\_scaled



2.1. Stochastic Modelling

#Multicollinearity Examination

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

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

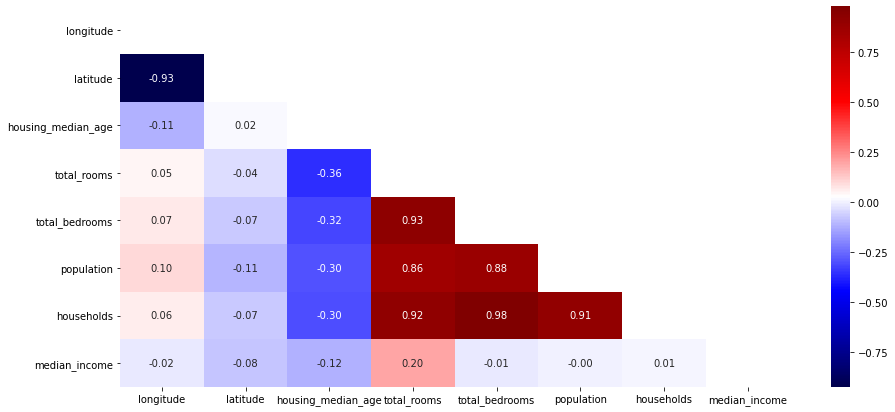
    mask = np.zeros\_like(x.corr(), dtype=np.bool)

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

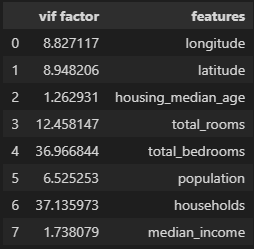
    sns.heatmap(x.corr(), annot=True, fmt='.2f', mask=mask, cmap='seismic')

    plt.show()

visualCorr(features\_scaled)

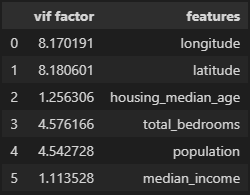


vif(features\_scaled)



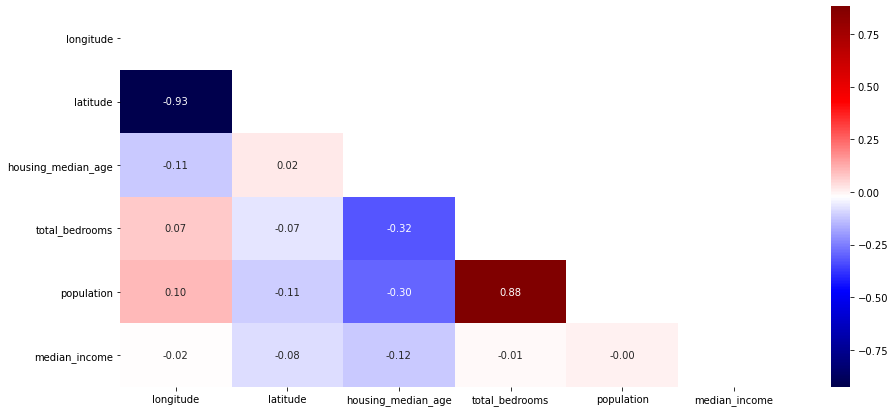
#Multicollinearity Resolution (VIF)

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

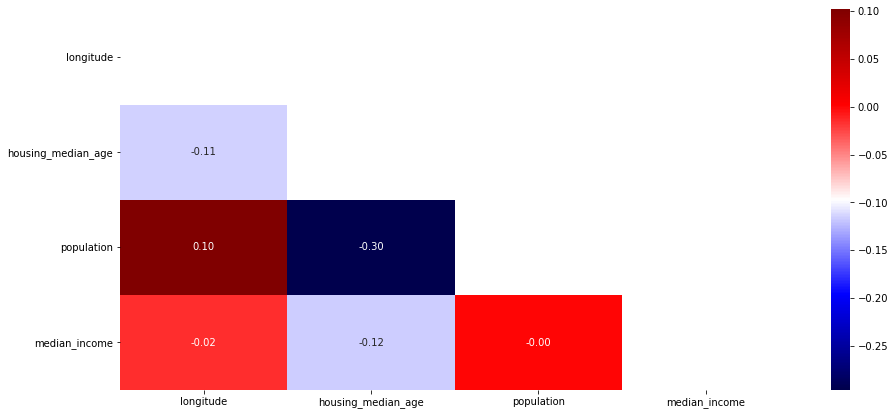


#Multicolinearity Resolution (Correlation)

visualCorr(features.drop(columns=['households','total\_rooms']))



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



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

features\_multicol



#OLS

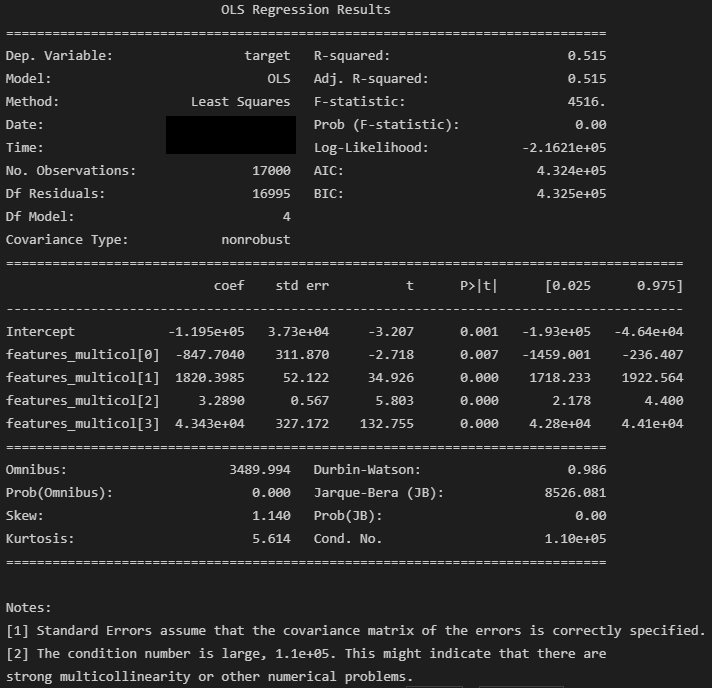
from statsmodels.formula.api import ols

target\_df = pd.DataFrame(data = target)

data\_all\_multicol = pd.concat([features\_multicol, target\_df], axis=1)

olsModel = ols('target ~ features\_multicol', data = data\_all\_multicol).fit()

print(olsModel.summary())



#OLS(2)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

target\_df = pd.DataFrame(target)

target\_scaled = target\_df.copy()

target\_scaled = scaler.fit\_transform(target\_df)

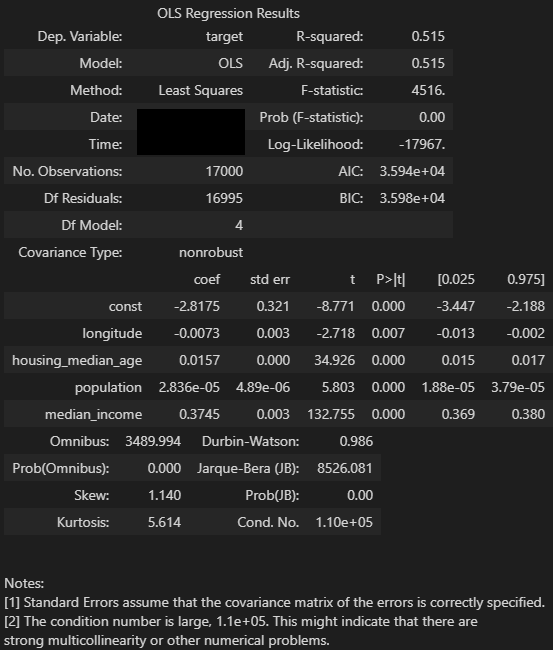
target\_scaled = pd.DataFrame(target\_scaled, columns=['target'])

import statsmodels.api as sm

X = sm.add\_constant(features\_multicol)

targetscaledModel = sm.OLS(target\_scaled, X).fit()

targetscaledModel.summary()



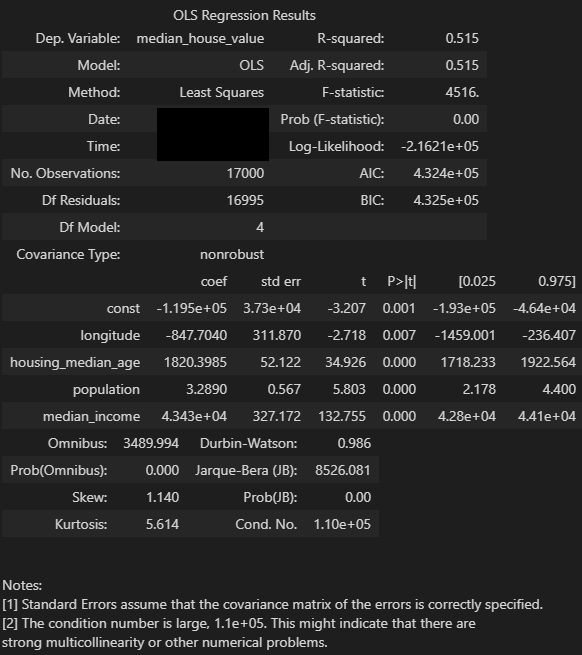
#OLS(3)

import statsmodels.api as sm

X = sm.add\_constant(features\_multicol)

model = sm.OLS(target, X).fit()

model.summary()



It can be inferred from the OLS summary that all variables rescaled and selected via multicollinearity resolution process are statistically significant (p<0.01), and the variables longitude and housing median age are the particularly impactful variables in terms of coefficients (-847 and 1820, respectively).

3. Feature Selection Comparisons (Scaled) and Evaluations

3.1. Forward Selection

def forward(model, x, y, col\_selected):

    col\_forw = [ cl for cl in features.columns if cl not in col\_selected ]

    rst = []

    for col in col\_forw:

        columns = col\_selected + [col]

        model = model.fit(x[columns], y)

        yhat = model.predict(x[columns])

        score = rSquare(x[columns], y, yhat)

        rst.append( {'model': model, 'score': score['adjr2'], 'columns': columns} )

    models = pd.DataFrame(rst)

    model\_best = models.loc[ models.score.argmax() ]

    return model\_best

def forward\_selection(x, y):

    col\_selected = []

    for i in range(0, x.shape[1]):

        model = LinearRegression()

        model\_best = forward(model, features, target, col\_selected)

        if not i:

            model\_prior = model\_best

        else:

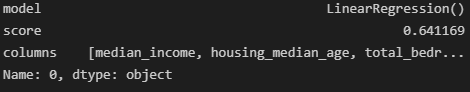
            if model\_best.score > model\_prior.score: model\_prior = model\_best

            else: break

        col\_selected = model\_best.columns

    return model\_prior

forward\_selection(features\_scaled, target)



3.2. Backward Elimination

from itertools import combinations

def backward(model, x, y, col\_selected):

    rst = []

    for com in combinations(col\_selected, len(col\_selected)-1):

        columns = list(com)

        model = model.fit(x[columns], y)

        yhat = model.predict(x[columns])

        score = rSquare(x[columns], y, yhat)

        rst.append( {'model': model, 'score': score['adjr2'], 'columns': columns} )

    models = pd.DataFrame(rst)

    model\_best = models.loc[ models.score.argmax() ]

    return model\_best

def backward\_elimination(x, y):

    col\_selected = x.columns

    for i in range(0, x.shape[1]):

        model = LinearRegression()

        model\_best = backward(model, features, target, col\_selected)

        if not i:

            model\_prior = model\_best

        else:

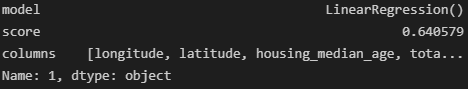
            if model\_best.score > model\_prior.score: model\_prior = model\_best

            else: break

        col\_selected = model\_best.columns

    return model\_prior

backward\_elimination(features\_selected, target)



3.3. Stepwise Selection

def forward(model, x, y, col\_selected):

    col\_forw = [ cl for cl in features.columns if cl not in col\_selected ]

    rst = []

    for col in col\_forw:

        columns = col\_selected + [col]

        model = model.fit(x[columns], y)

        yhat = model.predict(x[columns])

        score = rSquare(x[columns], y, yhat)

        rst.append( {'model': model, 'score': score['adjr2'], 'columns': columns} )

    models = pd.DataFrame(rst)

    model\_best = models.loc[ models.score.argmax() ]

    return model\_best

def backward(model, x, y, col\_selected):

    rst = []

    for com in combinations(col\_selected, len(col\_selected)-1):

        columns = list(com)

        model = model.fit(x[columns], y)

        yhat = model.predict(x[columns])

        score = rSquare(x[columns], y, yhat)

        rst.append( {'model': model, 'score': score['adjr2'], 'columns': columns} )

    models = pd.DataFrame(rst)

    model\_best = models.loc[ models.score.argmax() ]

    return model\_best

def stepwise\_selection(x, y):

    col\_selected = []

    for i in range(x.shape[1]):

        model = LinearRegression()

        model\_forw = forward(model, x, y, col\_selected)

        col\_selected = model\_forw.columns

        if i < 1: model\_prior = model\_forw; continue

        model\_back = backward(model, x, y, col\_selected)

        model\_high = model\_forw

        if model\_forw.score < model\_back.score:

            col\_selected = model\_back.columns

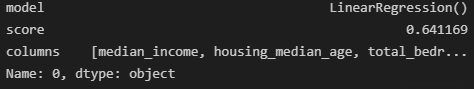
            model\_high = model\_back

        if model\_high.score > model\_prior.score: model\_prior = model\_high

        else: break

    return model\_prior

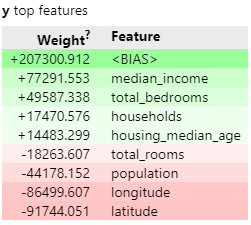
stepwise\_selection(features\_selected, target)



#Feature Weights Evaluation (1)

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

eli5.show\_weights(model, feature\_names = features\_scaled.columns.tolist())

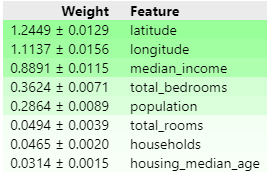


#Feature Weights Evaluation (2)

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

perm = PermutationImportance(model).fit(features\_scaled, target)

eli5.show\_weights(perm, feature\_names = features\_scaled.columns.tolist())



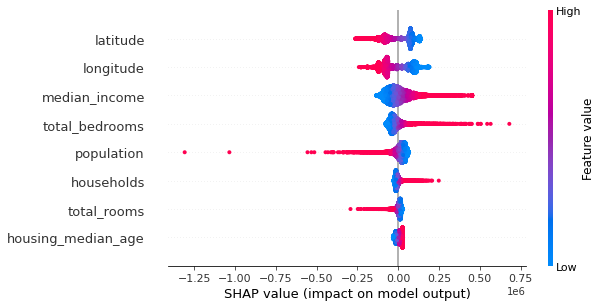
#Feature Weights Evaluation (3)

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

explainer = shap.LinearExplainer(model, features\_scaled)

shap\_value = explainer.shap\_values(features\_scaled)

shap.summary\_plot(shap\_value, features\_scaled)



#Feature Selection (RFE)

from sklearn.feature\_selection import RFE

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

feature\_selector = RFE(model, n\_features\_to\_select=4)

feature\_selector = feature\_selector.fit(features\_scaled, target)

display(feature\_selector.n\_features\_)

display(feature\_selector.support\_)

features\_scaled.loc[:, feature\_selector.support\_]

