Untitled

May 8, 2024

[19]:
```
# 1

import pandas as pd

variable_description_df = pd.read_excel('/Users/rouren/Desktop/24S ML/HW/mini3/
↪PPHA_30546_MP03-Variable_Description.xlsx')

variables_to_keep = variable_description_df.iloc[2:62, 1].tolist() +␣
↪['deathspc'] + ['county'] + ['state']

data_covid_df = pd.read_csv('/Users/rouren/Desktop/24S ML/HW/mini3/
↪Data-Covid002.csv', encoding='latin1')  # Assuming it's a CSV file, adjust␣
↪accordingly if it's in a different format

filtered_data_covid_df = data_covid_df[variables_to_keep]
```

[20]:
```
# 2

subset_df = data_covid_df[variables_to_keep]

summary_statistics = subset_df.describe()

print(summary_statistics)
```

```
       intersects_msa  cur_smoke_q1  cur_smoke_q2  cur_smoke_q3  cur_smoke_q4  \
count     3107.000000   3107.000000   3107.000000   3107.000000   3107.000000
mean         0.596717      0.212659      0.171048      0.134467      0.098316
std          0.490636      0.149348      0.128130      0.132181      0.110110
min          0.000000      0.000000      0.000000      0.000000      0.000000
25%          0.000000      0.000000      0.000000      0.000000      0.000000
50%          1.000000      0.250000      0.198718      0.142857      0.096535
75%          1.000000      0.310931      0.250000      0.200000      0.148719
max          1.000000      1.000000      1.000000      1.000000      1.000000

       bmi_obese_q1  bmi_obese_q2  bmi_obese_q3  bmi_obese_q4  \
count   3107.000000   3107.000000   3107.000000   3107.000000
mean       0.239166      0.214580      0.209621      0.186739
std        0.165928      0.153237      0.175849      0.167227
```

```
min        0.000000      0.000000      0.000000      0.000000
25%        0.080128      0.000000      0.000000      0.000000
50%        0.272076      0.241590      0.223124      0.194118
75%        0.335532      0.304348      0.297220      0.266667
max        1.000000      1.000000      1.000000      1.000000

       exercise_any_q1  …      taxrate  tax_st_diff_top20         pm25  \
count      3107.000000  …  3107.000000        3106.000000  3107.000000
mean          0.455995  …     0.023089           0.775634     8.371871
std           0.273874  …     0.013848           1.470989     2.565927
min           0.000000  …     0.000000           0.000000     0.000000
25%           0.312500  …     0.014993           0.000000     6.309710
50%           0.566563  …     0.020339           0.000000     8.784647
75%           0.641509  …     0.027164           1.000000    10.483764
max           1.000000  …     0.209907           7.220000    15.786018

          pm25_mia  summer_tmmx  summer_rmax  winter_tmmx  winter_rmax  \
count  3107.000000  3107.000000  3107.000000  3107.000000  3107.000000
mean      0.003540   303.126997    88.970517   280.404875    87.469432
std       0.059405     3.173950     9.689271     6.597855     4.811207
min       0.000000   290.455540    31.643282   264.693820    58.159798
25%       0.000000   300.848035    88.052494   275.113020    85.093342
50%       0.000000   303.290440    91.320313   280.154690    88.028793
75%       0.000000   305.817430    94.812389   285.543750    90.747704
max       1.000000   313.872680    99.778748   298.340360    97.672874

       bmcruderate     deathspc
count   3107.00000  3107.000000
mean    1029.15597    23.790131
std      248.38181    67.852145
min      189.30000     0.000000
25%      864.29999     0.000000
50%     1036.30000     3.802303
75%     1194.10000    21.461759
max     1978.60000  2279.610600

[8 rows x 61 columns]
```

[21]:
```python
# 3

# Drop all observations with missing values
subset_df = subset_df.dropna()
```

[22]:
```python
# 4

state_dummies = pd.get_dummies(subset_df['state'])
```

```
subset_df = pd.concat([subset_df, state_dummies], axis=1)

subset_df = subset_df.drop(columns=['state'])
```

[23]:

```
# 5

from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(subset_df, test_size=0.2,␣
 ↪random_state=11)
```

[24]:

```
# 6

# (a)
import statsmodels.api as sm

X_train = train_set.drop('deathspc', axis=1)
y_train = train_set['deathspc']
X_test = test_set.drop('deathspc', axis=1)
y_test = test_set['deathspc']
X_train.drop('county', axis=1, inplace=True)
X_test.drop('county', axis=1, inplace=True)

non_numeric_columns = X_train.select_dtypes(exclude=['int64', 'float64']).
 ↪columns

print("Non-numeric columns:", non_numeric_columns)

# Convert dummy variables to integers
for state in non_numeric_columns[1:]:
    X_train[state] = X_train[state].astype(int)
    X_test[state] = X_test[state].astype(int)


X_train.dropna(inplace=True)
y_train = y_train.loc[X_train.index]
X_test.dropna(inplace=True)
y_test = y_test.loc[X_test.index]

X_train_sm = sm.add_constant(X_train)
X_test_sm = sm.add_constant(X_test)

# Fit the OLS model
model = sm.OLS(y_train, X_train_sm).fit()

y_train_pred = model.predict(X_train_sm)
y_test_pred = model.predict(X_test_sm)
```

```
mse_train = ((y_train - y_train_pred) ** 2).mean()
mse_test = ((y_test - y_test_pred) ** 2).mean()

print(f"Training MSE: {mse_train}")
print(f"Test MSE: {mse_test}")
```

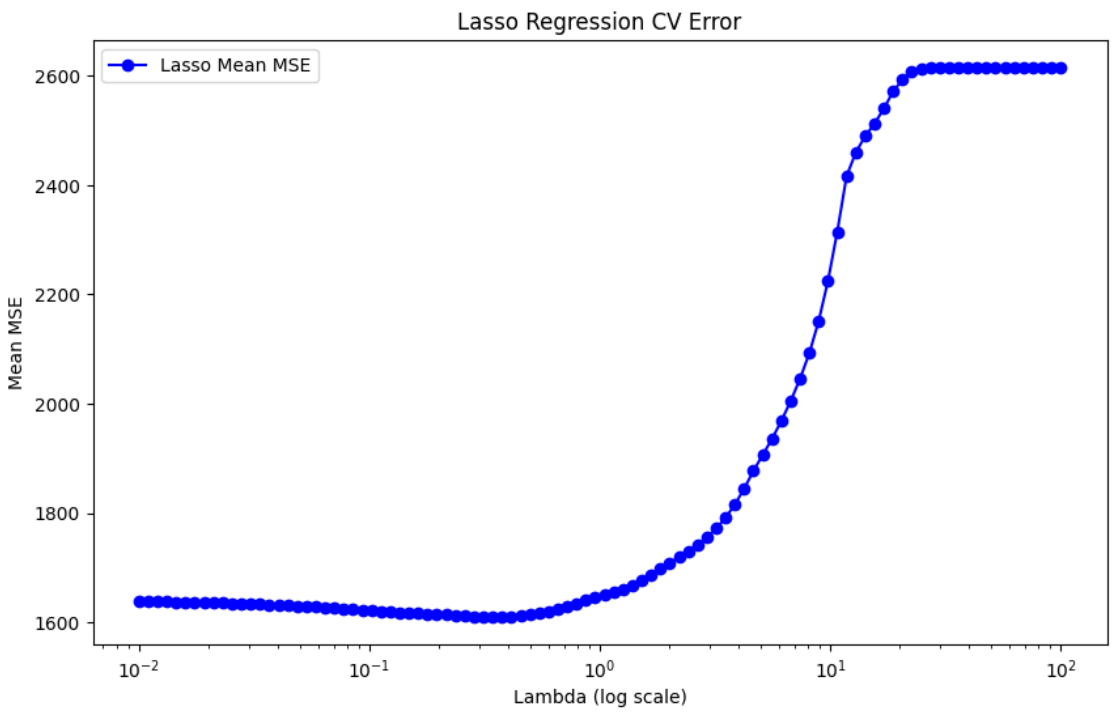
```
Non-numeric columns: Index(['Alabama', 'Arizona', 'Arkansas', 'California',
'Colorado',
       'Connecticut', 'Delaware', 'Florida', 'Georgia', 'Idaho', 'Illinois',
       'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Maine',
       'Maryland', 'Massachusetts', 'Michigan', 'Minnesota', 'Mississippi',
       'Missouri', 'Montana', 'Nebraska', 'Nevada', 'New Hampshire',
       'New Mexico', 'New York', 'North Carolina', 'North Dakota', 'Ohio',
       'Oklahoma', 'Oregon', 'Pennsylvania', 'Rhode Island', 'South Carolina',
       'South Dakota', 'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia',
       'Washington', 'West Virginia', 'Wisconsin', 'Wyoming'],
      dtype='object')
Training MSE: 1440.7028993710019
Test MSE: 2408.747453859704
```

[25]:
```
# (b)

# Overfitting in this model is likely driven by a high number of predictors,
↪including state dummies, which can cause the model to overly fit the nuances
↪of the training data and impair its generalizability. This is evidenced by
↪the higher MSE on the test set compared to the training set, indicating that
↪the model captures noise and non-generalizable patterns from the training
↪data. To enhance predictive accuracy and reliability, consider simplifying
↪the model, applying regularization, or using robust cross-validation methods.
```

[26]:
```
# 7

# (a)&(b)

import numpy as np
import pandas as pd
from sklearn.linear_model import RidgeCV, LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

lambdas = np.logspace(-2, 2, 100)
```

```
kf = KFold(n_splits=10, shuffle=True, random_state=25)

ridge_cv = RidgeCV(alphas=lambdas, cv=kf, scoring='neg_mean_squared_error')
lasso_cv = LassoCV(alphas=lambdas, cv=kf, random_state=25)

ridge_cv.fit(X_train_scaled, y_train)
lasso_cv.fit(X_train_scaled, y_train)

# (c)

plt.figure(figsize=(10, 6))
plt.plot(lasso_cv.alphas_, lasso_cv.mse_path_.mean(axis=1), marker='o',
 ↪linestyle='-', color='b', label='Lasso Mean MSE')
plt.xscale('log')
plt.xlabel('Lambda (log scale)')
plt.ylabel('Mean MSE')
plt.title('Lasso Regression CV Error')
plt.legend()
plt.show()


# (d)

print("Optimal lambda for Ridge:", ridge_cv.alpha_)
print("Optimal lambda for Lasso:", lasso_cv.alpha_)

# (e)

from sklearn.linear_model import Ridge, Lasso

optimal_ridge = Ridge(alpha=ridge_cv.alpha_)
optimal_lasso = Lasso(alpha=lasso_cv.alpha_)

optimal_ridge.fit(X_train_scaled, y_train)
optimal_lasso.fit(X_train_scaled, y_train)


# 8

from sklearn.metrics import mean_squared_error

y_train_pred_ridge = optimal_ridge.predict(X_train_scaled)
y_train_pred_lasso = optimal_lasso.predict(X_train_scaled)

y_test_pred_ridge = optimal_ridge.predict(X_test_scaled)
y_test_pred_lasso = optimal_lasso.predict(X_test_scaled)
```

```
mse_train_ridge = mean_squared_error(y_train, y_train_pred_ridge)
mse_train_lasso = mean_squared_error(y_train, y_train_pred_lasso)

mse_test_ridge = mean_squared_error(y_test, y_test_pred_ridge)
mse_test_lasso = mean_squared_error(y_test, y_test_pred_lasso)

print(f"Ridge Regression Training MSE: {mse_train_ridge}")
print(f"Lasso Regression Training MSE: {mse_train_lasso}")
print(f"Ridge Regression Test MSE: {mse_test_ridge}")
print(f"Lasso Regression Test MSE: {mse_test_lasso}")
```




```
Optimal lambda for Ridge: 100.0
Optimal lambda for Lasso: 0.34304692863149194
Ridge Regression Training MSE: 1449.9762401047685
Lasso Regression Training MSE: 1471.4957284887846
Ridge Regression Test MSE: 2404.5057732692303
Lasso Regression Test MSE: 2396.7314112860718
```

[27]:
```
# Both Ridge and Lasso regression improved prediction over the OLS model, with␣
↪Lasso showing the most significant improvement. Lasso Regression had the␣
↪lowest test MSE, indicating better generalization to unseen data.
```

```
# I recommend Lasso Regression to the CDC because it not only minimizes␣
↪overfitting but also simplifies the model by reducing the number of␣
↪features, which aids in interpretability. This is crucial for making␣
↪informed public health decisions based on key predictive factors.
```