Mini-Project-3

Daniel Avila

3.

(1)

```
import pandas as pd

import numpy as np
import matplotlib.pyplot as plt
import os
import statsmodels.api as sms
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV

os.chdir("C:/Users/danie/Documents/GitHub/Machine-Learning--Harris/Mini-Project-3")

covid_df = pd.read_csv("Data-Covid003.csv", encoding = 'latin1')

var_des = pd.read_excel("PPHA_30545_MP03-Variable_Description.xlsx")
var_list = list(var_des["Variable"])
var_list.append("county")
var_list.append("state")
var_list.remove("casespc")

covid_df = covid_df[covid_df.columns[covid_df.columns.isin(var_list)]]
```

(2)

```
pd.set_option("display.max_rows", None)
summary_stats = covid_df.describe().T
summary_stats = summary_stats.loc[:, ["count", "mean", "std"]]
summary_stats
```

	count	mean	std
deathspc	3107.0	0.751995	1.792700
intersects_msa	3107.0	0.596717	0.490636
cur_smoke_q1	3107.0	0.212659	0.149348
cur_smoke_q2	3107.0	0.171048	0.128130
cur_smoke_q3	3107.0	0.134467	0.132181
cur_smoke_q4	3107.0	0.098316	0.110110
bmi_obese_q1	3107.0	0.239166	0.165928
bmi_obese_q2	3107.0	0.214580	0.153237
bmi_obese_q3	3107.0	0.209621	0.175849
bmi_obese_q4	3107.0	0.186739	0.167227
exercise_any_q1	3107.0	0.455995	0.273874
exercise_any_q2	3107.0	0.555671	0.322336
exercise_any_q3	3107.0	0.603792	0.357861
exercise_any_q4	3107.0	0.638727	0.376922
brfss_mia	3107.0	0.249437	0.432757
puninsured2010	3107.0	18.469460	5.536651
reimb_penroll_adj10	3103.0	9302.737743	1590.926253
mort_30day_hosp_z	3106.0	0.457806	1.206493
adjmortmeas_amiall30day	3106.0	0.165483	0.039408
adjmortmeas_chfall30day	3107.0	0.108969	0.023565
med_prev_qual_z	3012.0	-0.148547	0.863881
primcarevis_10	3098.0	80.865348	7.401457
diab_hemotest_10	3069.0	83.706025	6.594153
diab_eyeexam_10	3054.0	66.080221	7.598549
diab_lipids_10	3057.0	78.307420	7.854145
mammogram_10	3029.0	63.110073	8.397699
cs00_seg_inc	3107.0	0.025892	0.030576
cs00_seg_inc_pov25	3107.0	0.024278	0.030757
cs00_seg_inc_aff75	3107.0	0.026463	0.032920
cs_race_theil_2000	3107.0	0.075402	0.084131
gini99	3008.0	0.379021	0.086677
poor_share	3107.0	0.141739	0.065460
inc_share_1perc	3008.0	0.094808	0.050631
frac_middleclass	3106.0	0.554244	0.093099
scap_ski90pcm	3107.0	0.000182	1.347960

	count	mean	std
rel_tot	3106.0	53.224564	18.502524
cs_frac_black	3107.0	8.744503	14.483719
cs_frac_hisp	3107.0	6.209190	12.050404
unemp_rate	3107.0	0.049871	0.017738
cs_labforce	3107.0	0.609344	0.070393
cs_elf_ind_man	3107.0	0.159118	0.090862
cs_born_foreign	3107.0	3.441958	4.836270
mig_inflow	3017.0	0.028677	0.019034
mig_outflow	3017.0	0.027522	0.013780
pop_density	3107.0	244.325026	1676.096088
frac_traveltime_lt15	3107.0	0.403803	0.137215
hhinc00	3107.0	32853.502978	6975.837500
median_house_value	3107.0	112180.080571	63189.048357
ccd_exp_tot	3080.0	6.092697	2.103573
score_r	3069.0	0.077348	9.007980
cs_fam_wkidsinglemom	3107.0	0.194598	0.067828
subcty_exp_pc	3107.0	2119.407531	999.833466
taxrate	3107.0	0.023089	0.013848
tax_st_diff_top20	3106.0	0.775634	1.470989
summer_tmmx	3107.0	303.126997	3.173950
summer_rmax	3107.0	88.970517	9.689271
winter_tmmx	3107.0	280.404875	6.597855
winter_rmax	3107.0	87.469432	4.811207
pm25	3107.0	8.371871	2.565927
bmcruderate	3107.0	1029.155970	248.381810
pm25_mia	3107.0	0.003540	0.059405

(3)

```
missing_values = pd.DataFrame(np.sum(covid_df.isna(), axis = 0), columns = ["NA's"])
print(f"there are some columns with missing values: {missing_values["NA's"].to_markdown()}")

covid_df = covid_df.dropna()
```

```
there are some columns with missing values: |                          |   NA's |
|:-------------------------|-------:|
| county                   |      0 |
| state                    |      0 |
| deathspc                 |      0 |
| intersects_msa           |      0 |
```

cur_smoke_q1	0
cur_smoke_q2	0
cur_smoke_q3	0
cur_smoke_q4	0
bmi_obese_q1	0
bmi_obese_q2	0
bmi_obese_q3	0
bmi_obese_q4	0
exercise_any_q1	0
exercise_any_q2	0
exercise_any_q3	0
exercise_any_q4	0
brfss_mia	0
puninsured2010	0
reimb_penroll_adj10	4
mort_30day_hosp_z	1
adjmortmeas_amiall30day	1
adjmortmeas_chfall30day	0
med_prev_qual_z	95
primcarevis_10	9
diab_hemotest_10	38
diab_eyeexam_10	53
diab_lipids_10	50
mammogram_10	78
cs00_seg_inc	0
cs00_seg_inc_pov25	0
cs00_seg_inc_aff75	0
cs_race_theil_2000	0
gini99	99
poor_share	0
inc_share_1perc	99
frac_middleclass	1
scap_ski90pcm	0
rel_tot	1
cs_frac_black	0
cs_frac_hisp	0
unemp_rate	0
cs_labforce	0
cs_elf_ind_man	0
cs_born_foreign	0
mig_inflow	90
mig_outflow	90
pop_density	0

```
| frac_traveltime_lt15      |       0 |
| hhinc00                   |       0 |
| median_house_value        |       0 |
| ccd_exp_tot               |      27 |
| score_r                   |      38 |
| cs_fam_wkidsinglemom      |       0 |
| subcty_exp_pc             |       0 |
| taxrate                   |       0 |
| tax_st_diff_top20         |       1 |
| summer_tmmx               |       0 |
| summer_rmax               |       0 |
| winter_tmmx               |       0 |
| winter_rmax               |       0 |
| pm25                      |       0 |
| bmcruderate               |       0 |
| pm25_mia                  |       0 |
```

(4)

```python
for state in covid_df["state"].unique():
    covid_df[state] = np.where(covid_df["state"] == state, 1, 0)
```

(5)

```python
from sklearn.model_selection import train_test_split

X = covid_df.loc[:, ~covid_df.columns.isin(["deathspc", "county", "state"])]
y = covid_df.loc[:, "deathspc"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .2, random_state = 11)
```

(6)

(a)

```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

model = LinearRegression()
model.fit(X_train, y_train)

y_pred_train = model.predict(X_train)
mse_train = round(np.mean((y_train - y_pred_train)**2), 2)
```

```
y_pred_test = model.predict(X_test)
mse_test = round(np.mean((y_test - y_pred_test)**2), 2)

print(f"MSE train: {mse_train}")
print(f"MSE test: {mse_test}")
```

```
MSE train: 1.29
MSE test: 1.83
```

(b) There is potential for overfitting because, given the large number of variables (112) that we are using in our dataset, each variable is treated by the model equally in terms of prediction power. Therefore the variables that actually have low prediction power would over-influence our predcitions, which would make us overfit to the training set.

We also have evidence that we might be overfitting, as the training MSE and the test MSE demonstrate a nearly ~41% difference. This would indicate that our OLS model is doing well in fitting training data, but might not be doing as well as possible to new data.

(7) (a), (b)

```
#setting up the grid search for hyperparameters
from sklearn.preprocessing import StandardScaler

lasso = Lasso()
ridge = Ridge()

scaler= StandardScaler()
scaler.fit(X_train)
X_train=scaler.transform(X_train)
X_test= scaler.transform(X_test)

alpha_param = np.power(10, (np.linspace(-2, 1, 100)))

grid_search_lasso = GridSearchCV(lasso, alpha_param)
grid_search_ridge = GridSearchCV(ridge, alpha_param)

#Creating a parameters grid
param_grid = [{
    'alpha': alpha_param
}]

#running this for lasso first
```

```
#Running Grid Search over the alpha (regularization) parameter
kfcv = KFold(n_splits=10, random_state = 25, shuffle=True)
grid_search_lasso = GridSearchCV(lasso, param_grid, cv=kfcv, scoring='neg_mean_squared_error
grid_search_lasso.fit(X_train, y_train)

# Extract results for all tested alphas
tested_alphas = []
mean_vec_lasso = []
std_test_score = []

for params in grid_search_lasso.cv_results_["params"]:
    tested_alphas.append(params['alpha'])

for mse in grid_search_lasso.cv_results_["mean_test_score"]:
    mean_vec_lasso.append(-mse)

for std in grid_search_lasso.cv_results_["std_test_score"]:
    std_test_score.append(std)


# Store mean and standard deviation values
results_cv_lasso = pd.DataFrame({'alpha': tested_alphas, 'MSE': mean_vec_lasso, "STD": std_te

#now ridge

grid_search_ridge = GridSearchCV(ridge, param_grid, cv=kfcv, scoring='neg_mean_squared_error
grid_search_ridge.fit(X_train, y_train)

# Extract results for all tested alphas
tested_alphas = []
mean_vec_ridge = []
std_test_score = []

for params in grid_search_ridge.cv_results_["params"]:
    tested_alphas.append(params['alpha'])

for mse in grid_search_ridge.cv_results_["mean_test_score"]:
    mean_vec_ridge.append(-mse)

for std in grid_search_ridge.cv_results_["std_test_score"]:
    std_test_score.append(std)
```

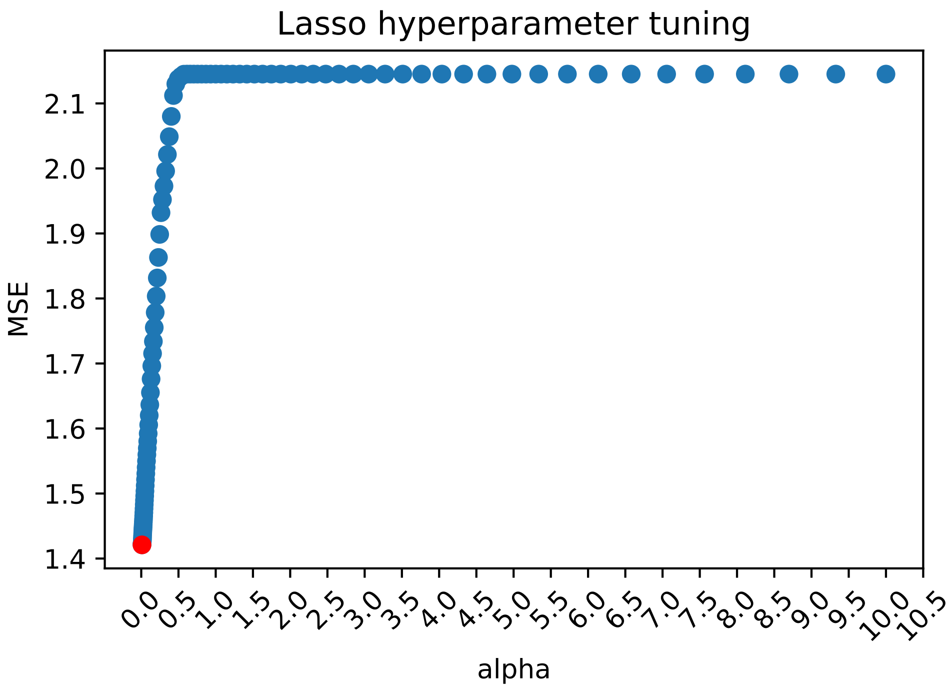
```
# Store mean and standard deviation values
results_cv_ridge = pd.DataFrame({'alpha': tested_alphas, 'MSE': mean_vec_ridge, "STD": std_te
```

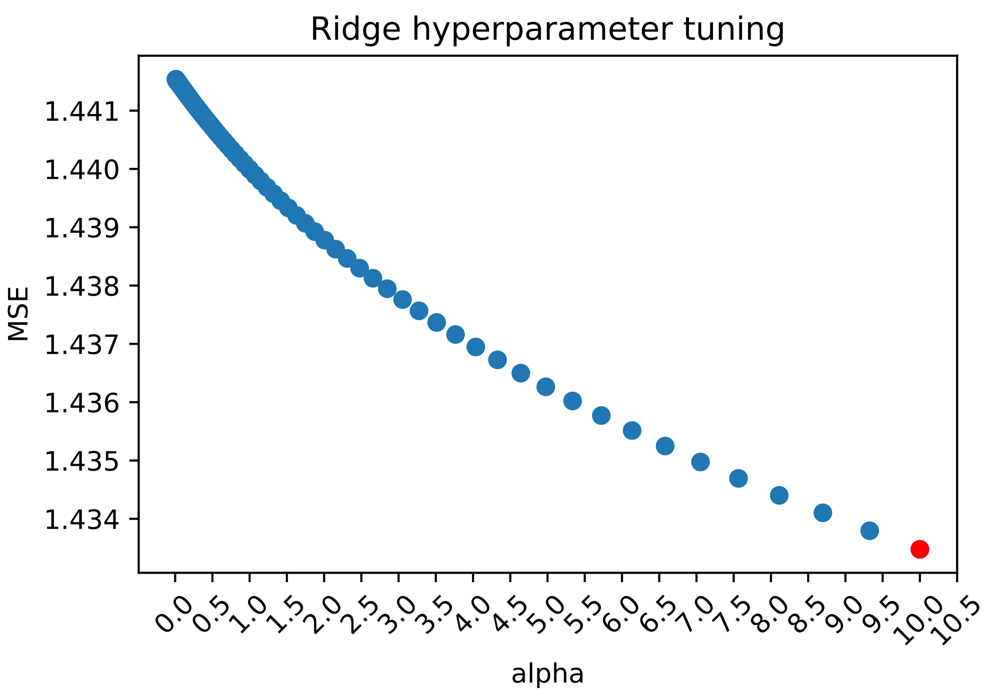
(c)

```
min_mse_ridge = results_cv_ridge[results_cv_ridge["MSE"] == results_cv_ridge["MSE"].min()]
min_mse_lasso = results_cv_lasso[results_cv_lasso["MSE"] == results_cv_lasso["MSE"].min()]

#lasso plot
plt.scatter(results_cv_lasso["alpha"], results_cv_lasso["MSE"])
plt.scatter(min_mse_lasso["alpha"], min_mse_lasso["MSE"], color = "red")
plt.title("Lasso hyperparameter tuning")
plt.xlabel("alpha")
plt.xticks((np.arange(0, results_cv_lasso["alpha"].max() + 1, .5)), rotation = 45)
plt.ylabel("MSE")
plt.show()

#ridge plot
plt.scatter(results_cv_ridge["alpha"], results_cv_ridge["MSE"])
plt.scatter(min_mse_ridge["alpha"], min_mse_ridge["MSE"], color = "red")
plt.title("Ridge hyperparameter tuning")
plt.xlabel('alpha')
plt.xticks((np.arange(0, results_cv_ridge["alpha"].max() + 1, .5)), rotation = 45)
plt.ylabel("MSE")
plt.show()
```


Lasso hyperparameter tuning
MSE
2.1
2.0
1.9
1.8
1.7
1.6
1.5
1.4
alpha
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0 10.5


Ridge hyperparameter tuning
MSE
1.441
1.440
1.439
1.438
1.437
1.436
1.435
1.434
alpha
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0 10.5

(d)

```
print(f"min MSE and given alpha, ridge:")
print(min_mse_ridge)
print()
print(f"min MSE and given alpha, lasso:")
print(min_mse_lasso)
```

```
min MSE and given alpha, ridge:
    alpha       MSE       STD
99   10.0  1.433477  0.392839

min MSE and given alpha, lasso:
   alpha       MSE       STD
0   0.01  1.421054  0.407232
```

(e)

```
#training lasso first
lasso = Lasso(alpha = min_mse_lasso.iloc[0, 0])
lasso.fit(X_train, y_train)
lasso_pred = lasso.predict(X_test)
lasso_mse = mean_squared_error(y_test, lasso_pred)

lasso_pred_train = lasso.predict(X_train)
lasso_mse_train = mean_squared_error(y_train, lasso_pred_train)

#ridge next
ridge = Ridge(alpha = min_mse_ridge.iloc[0, 0])
ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)
ridge_mse = mean_squared_error(y_test, ridge_pred)

ridge_pred_train = ridge.predict(X_train)
ridge_mse_train = mean_squared_error(y_train, ridge_pred_train)
```

(8)

```
print(f"train lasso mse: {round(lasso_mse_train, 3)}")
print(f"optimized test lasso mse: {round(lasso_mse, 3)}")
print()
print(f"train ridge mse: {round(ridge_mse_train, 3)}")
print(f"optimized test ridge mse: {round(ridge_mse, 3)}")
print()
```

```python
print(f"train OLS mse: {round(mse_train, 2)}")
print(f"optimized test OLS mse: {round(mse_test, 3)}")
```

```
train lasso mse: 1.31
optimized test lasso mse: 1.808

train ridge mse: 1.287
optimized test ridge mse: 1.826

train OLS mse: 1.29
optimized test OLS mse: 1.83
```

It seems that the lasso and ridge methods do not provide a significant amount of benefit relative to the OLS model that we estimated in the training set. However, ridge does seem to improve in terms of MSE when using new data from the test set. However, in testing performance relative to the test set, we find that the Lasso method does better when ingesting new data. Because the lasso method can send some variables to zero, we see that there are some variablse within our model that do not provide us with any predictive power. In relation with the CDC, we would recommend using the lasso method since it works better in predicting out of sample values.