ML MP#3

February 22, 2023

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
%matplotlib inline
import seaborn as sns
sns.set()
sns.set_style("darkgrid")
import warnings
warnings.filterwarnings("ignore")
import math
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
import os
from sklearn.model_selection import train_test_split
import statsmodels.formula.api as smf
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing
import numpy as np
import pandas as pd
import statsmodels.api as sm
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

[ ]:
```
base_path = r'C:\Users\frank\OneDrive\Desktop\Machine Learning\MP#3'
path = os.path.join(base_path,'Data-Covid002.csv')
covid_data = pd.read_csv(path, encoding='ISO-8859-1')
path2 = os.path.join(base_path,'PPHA_30545_MP03-Variable_Description.xlsx')
data_dict = pd.read_excel(path2)
```

1 Question 1

[ ]:
```
opp_insights_variables = data_dict.loc[data_dict['Source'] == 'Opportunity␣
 ↪Insights', 'Variable']
pm_covid_variables = data_dict.loc[data_dict['Source'] == 'PM_COVID',␣
 ↪'Variable']
other_variables = ['county','state','deathspc']
```

[ ]:
```
opp_insights_df = covid_data[covid_data.columns.
 ↪intersection(opp_insights_variables)]
pm_covid_df = covid_data[covid_data.columns.intersection(pm_covid_variables)]
other_variables_df = covid_data[covid_data.columns.
 ↪intersection(other_variables)]
```

[ ]:
```
concat_list = [other_variables_df, opp_insights_df, pm_covid_df]
filtered_data = pd.concat(concat_list, axis='columns')
filtered_data.head(10)
```

Links used:

https://stackoverflow.com/questions/40636514/selecting-columns-by-list-and-columns-are-subset-of-list

https://www.statology.org/summary-statistics-pandas/

https://sparkbyexamples.com/pandas/pandas-extract-column-value-based-on-another-column/

Other methods:

https://www.statology.org/pandas-extract-column-value-based-on-another-column/

https://stackoverflow.com/questions/57695964/using-lists-in-a-pandas-query

2 Question 2

[ ]:
```
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

[ ]:
```
filtered_data.describe()
```

3 Question 3

[ ]:
```
filtered_data.info()
```

[ ]:
```
filtered_data.dropna(inplace=True)
```

[ ]:
```
filtered_data.info()
```

4 Question 4

[ ]:
```
filtered_data = pd.get_dummies(filtered_data, columns=['state'])
```

[ ]:
```
filtered_data.head()
```

5 Question 5

[ ]:
```
X = filtered_data.drop(['deathspc','county'], axis=1)
y = filtered_data['deathspc']
```

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2,
random_state=10)
```

6 Question 6(a)

[ ]:
```
scaler = StandardScaler()
scaler.fit(X_train)
```

[ ]:
```
X_norm = scaler.transform(X_train)
```

[ ]:
```
reg = LinearRegression().fit(X_norm, y_train)
```

[ ]:
```
print(reg.score(X_norm, y_train))
print(reg.score(X_test, y_test))
```

[ ]:
```
y_pred = reg.predict(X_test)
mean_squared_error(y_test, y_pred)
```

7 Question 6(b)

Yes there is concern for overfitting. This is due to how low the MSE is for the training set (0.43199780957930267) but it instantly becomes extremely high for the test set (1.7131906369459238e+26). While we do expect the MSE to be higher for the test set than the training set, in this situation, the training MSE is a fraction while the test MSE is over septillion. This huge increase is suspicious and points to possible overfitting.

Links used:

https://stackoverflow.com/questions/58740329/patsyerror-number-of-rows-mismatch-between-data-argument-and-column-statsmodel

https://stackoverflow.com/questions/29586323/how-to-retain-column-headers-of-data-frame-after-pre-processing-in-scikit-learn

8 Question 7(a)(b)(c)

[ ]:
```
# Lasso
kf = KFold(n_splits=10, random_state = 25, shuffle=True)
tkf = kf.split(X, y)
lasso = Lasso()
alphacl = (np.linspace(0, 1, 101))
alpha_lgrid = [{'alpha': alphacl}]
def vector_values(grid_search, trials):
    mean_vec = np.zeros(trials)
    std_vec = np.zeros(trials)
    i = 0
    final = grid_search.cv_results_


    for mean_score, std_score in zip(final["mean_test_score"],
final["std_test_score"]):
        mean_vec[i] = -mean_score
        std_vec[i] = std_score
        i = i+1
    return mean_vec, std_vec
grid_search_lasso = GridSearchCV(lasso, alpha_lgrid, cv = tkf, scoring =
'neg_mean_squared_error')
grid_search_lasso.fit(X, y)
mean_vec, std_vec = vector_values(grid_search_lasso, 101)
results_cv_lasso = pd.DataFrame(
    {
        "alphas": alphacl,
        "MSE": mean_vec,
    }
)
min_mse = min(results_cv['MSE'])
results_cv_lasso.loc[results_cv['MSE']==min_mse]
plt.plot(
    alphacl,
    mean_vec,
    linewidth=2,
)

# Ridge
kf = KFold(n_splits=10, random_state = 25, shuffle=True)
tkf = kf.split(X, y)
ridge = Ridge()
alphacr = (np.linspace(0, 25, 251))
alpha_rgrid = [{'alpha': alphacr}]
grid_search_ridge = GridSearchCV(ridge, alpha_rgrid, cv = tkf, scoring =
'neg_mean_squared_error')
```

```
grid_search_ridge.fit(X, y)
mean_vec, std_vec = vector_values(grid_search_ridge, 251)
results_cv_ridge = pd.DataFrame(
    {
        "alphas": alphacr,
        "MSE": mean_vec,
    }
)
min_mse = min(results_cv['MSE'])
results_cv_ridge.loc[results_cv['MSE']==min_mse]
plt.plot(
    alphacr,
    mean_vec,
    linewidth=2,
)
```

9 Question 7(d)

[ ]:
```
results_cv_lasso.loc[results_cv['MSE']==min_mse]
```

[ ]:
```
results_cv_ridge.loc[results_cv['MSE']==min_mse]
```

10 Question 7(e)

[ ]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_norm = scaler.transform(X_train)
```

[ ]:
```
ols_lasso = Lasso(alpha=0.23)
ols_lasso.fit(X_norm, y_train)
ols_ridge = Ridge(alpha=2.3)
ols_ridge.fit(X_norm, y_train)
```

11 Question 8

[ ]:
```
y_pred_lasso = ols_lasso.predict(X_test)
mean_squared_error(y_test, y_pred_lasso)
```

[ ]:
```
y_pred_ridge = ols_ridge.predict(X_test)
mean_squared_error(y_test, y_pred_ridge)
```

Both lasso and ridge lowered the MSE by over half of what it was with OLS with lasso performing the best.

To be perfectly honest, none of these models are performing well. This could be because of the random state we are using although it is unclear without redoing all of this analysis with multiple different random states. But if I was forced to choose, I would choose lasso. First off, since OLS estimates have such a high variance in this situation, ridge and lasso would be better off. While ridge tends to perform better when the outcome is a function of relatively many predictors and lasso tends is better when the outcome is a function of relatively few predictors, that is a general statement. We will always need to test this to determine which approach works better and when we did lasso performs significantly better.

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```