WineQuality_project

July 2, 2021

1 Prediction on Wine Quality Project

1.0.1 Python(NumPy, Pandas, matplotlib, sklearn, GridSearchCV), Machine Learning models(OLS Regression, Ridge Regression, Lasso Regression), Jupyter Notebook

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import mean_absolute_error as mae

def Insert_row_(row_number, df, row_value):
    df1 = df[0:row_number]
    df2 = df[row_number:]
    df1.loc[row_number]=row_value
    df_result = pd.concat([df1, df2])
    df_result.index = [*range(df_result.shape[0])]
    return df_result

def OSR2(model, X_test, y_test, y_train):
    y_pred = model.predict(X_test)
    SSE = np.sum((y_test - y_pred)**2)
    SST = np.sum((y_test - np.mean(y_train))**2)
    return (1 - SSE/SST)

data = pd.read_csv("winequality-red.csv")
data.head()
```

[1]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
```

```
3           11.2              0.28         0.56              1.9       0.075
4            7.4              0.70         0.00              1.9       0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

With a given dataset containing fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality of wines, using a python language, I assigned y as a quality which is a subjective variable of wine dataset and X as a dataset of the other objective variables of wine and built following three kinds of models (X vs y): OLS regression, ridge regression, and lasso regression. First of all, I split the dataset into 70% of the training set and 30% of the test set with an intercept.

2 1. OLS Regression Model

[2]:
```
cols = ["fixed acidity", "volatile acidity","citric acid","residual␣
 ↪sugar","chlorides","free sulfur dioxide","total sulfur␣
 ↪dioxide","density","pH","sulphates","alcohol"]
X = data[cols]
y = data['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
olsmodel = linear_model.LinearRegression(fit_intercept=True)
olsmodel.fit(X_train, y_train)
```

[2]:
```
LinearRegression()
```

For the OLS regression model, I built a linear regression model with fit_intercept = True and fit to training sets of X and y. I got a coefficient table for the OLS model but no constant. By using a function Insert_row_, I inserted a constant variable to index 0 of the coefficient table for the OLS model. Based on the model, I found the OSR^2, MSE, and MAE of OLS model through a function of OSR2, mean_squared_error, and mean_absolute_error from sklearn.metrics.

3 1.1 Table of Coefficients

[3]:
```
# OLS Regression, Table of Coefficients
OLS_coeftable=pd.DataFrame(columns=["variable", "coefficient"])
OLS_coeftable["variable"]=cols
OLS_coeftable["coefficient"]= olsmodel.coef_
OLS_intercept = ['constant', olsmodel.intercept_]
OLS_coeftable = Insert_row_(0, OLS_coeftable, OLS_intercept)
OLS_coeftable
```

```
/opt/conda/lib/python3.8/site-packages/pandas/core/indexing.py:691:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
```

[3]:

	variable	coefficient
0	constant	-13.055380
1	fixed acidity	-0.004665
2	volatile acidity	-1.040221
3	citric acid	-0.084460
4	residual sugar	0.014429
5	chlorides	-1.862015
6	free sulfur dioxide	0.005614
7	total sulfur dioxide	-0.003658
8	density	17.588025
9	pH	-0.574364
10	sulphates	0.748007
11	alcohol	0.326325

4 1.2 OSR^2, MSE, MAE

[4]:
```
# OLS OSR^2, MSE, MAE
ols_osr2=OSR2(olsmodel, X_test, y_test, y_train)

predict_train = olsmodel.predict(X_train)
predict_test = olsmodel.predict(X_test)

ols_train_mse = mse(y_train, predict_train)
ols_test_mse = mse(y_test, predict_test)

ols_train_mae = mae(y_train, predict_train)
ols_test_mae = mae(y_test, predict_test)
```

```
print(ols_osr2)
print(ols_train_mse, ols_test_mse)
print(ols_train_mae, ols_test_mae)
```

```
0.315624168556l2363
0.4039421535663661 0.454281662251604
0.49343810403519406 0.5200946856740435
```

5 2. Ridge Regression Model

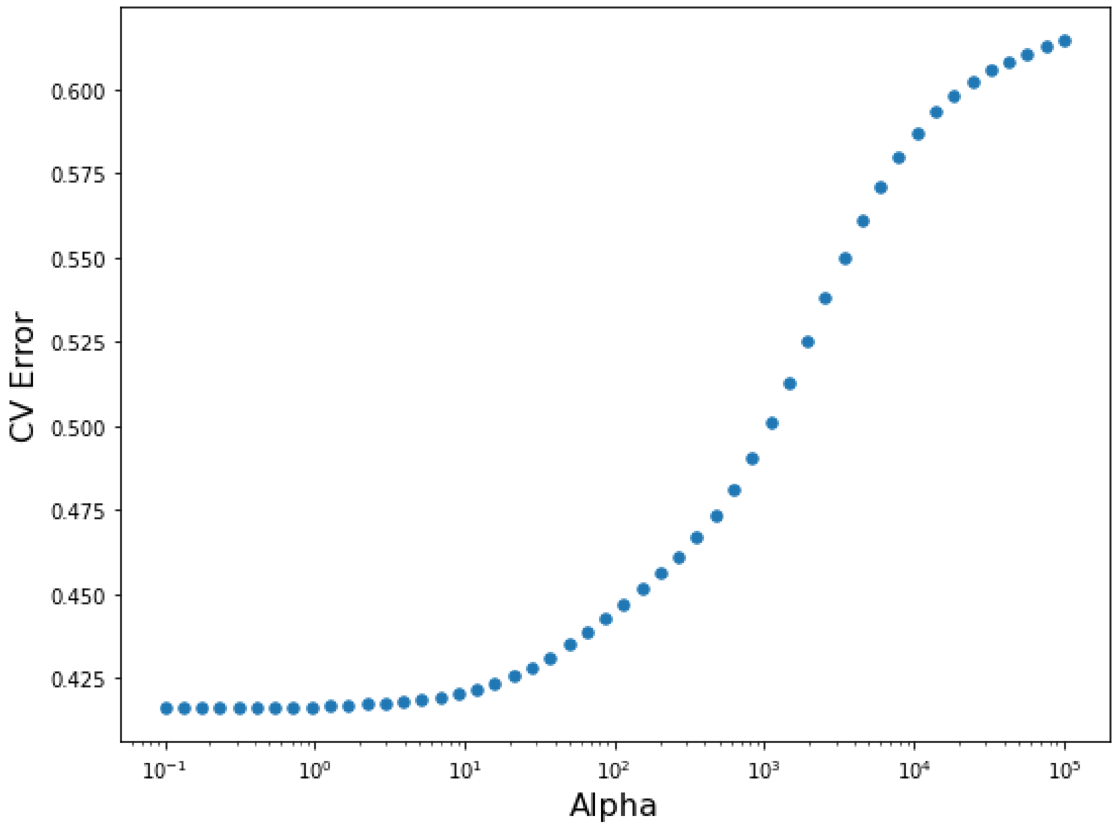
Since each ridge regression and lasso regression has one tuning parameter, for those two models, using GridSearchCV with 10 splits, I got a graph of CV error then found a best tuning parameter which results in the lowest CV error. With the best parameter, I made a table of coefficient for those two models. For the ridge model, I assigned an alpha_grid, 88 random states, and a scoring method as a negative mean squared error then fit to X and y. Later in order to find CV error score, I multiplied -1 since I used negative mean squared error. With those, I found a CV error graph of Ridge regression(log Alpha vs CV Error) model. I also found OSR2, MSE, and MAE for the ridge model.

[5]:
```
alpha_grid = {'alpha': np.logspace(-1, 5, num=50, base=10)}
rr = Ridge(random_state=88)
rr_cv = GridSearchCV(rr, alpha_grid, scoring='neg_mean_squared_error', cv=10)
rr_cv.fit(X_train, y_train)
```

[5]:
```
GridSearchCV(cv=10, estimator=Ridge(random_state=88),
             param_grid={'alpha': array([1.00000000e-01, 1.32571137e-01,
1.75751062e-01, 2.32995181e-01,
       3.08884360e-01, 4.09491506e-01, 5.42867544e-01, 7.19685673e-01,
       9.54095476e-01, 1.26485522e+00, 1.67683294e+00, 2.22299648e+00,
       2.94705170e+00, 3.90693994e+00, 5.17947468e+00, 6.86648845e+00,
       9.10298178e+00, 1.20679264e+01, 1…
       2.68269580e+02, 3.55648031e+02, 4.71486636e+02, 6.25055193e+02,
       8.28642773e+02, 1.09854114e+03, 1.45634848e+03, 1.93069773e+03,
       2.55954792e+03, 3.39322177e+03, 4.49843267e+03, 5.96362332e+03,
       7.90604321e+03, 1.04811313e+04, 1.38949549e+04, 1.84206997e+04,
       2.44205309e+04, 3.23745754e+04, 4.29193426e+04, 5.68986603e+04,
       7.54312006e+04, 1.00000000e+05])},
             scoring='neg_mean_squared_error')
```

[6]:
```
range_alpha = rr_cv.cv_results_['param_alpha'].data
CV_scores = rr_cv.cv_results_['mean_test_score']*(-1)
plt.figure(figsize=(8, 6))
ax = plt.gca()
ax.set_xscale('log')
plt.xlabel('Alpha', fontsize=16)
plt.ylabel('CV Error', fontsize=16)
plt.scatter(range_alpha, CV_scores, s=30)
```

```
plt.tight_layout()
plt.show()
```



[7]:
```
print(rr_cv.best_params_)
```

```
{'alpha': 0.2329951810515372}
```

6 2.1 Table of Coefficient

[8]:
```
real_ridge = Ridge(alpha=0.9540954763499939, fit_intercept=True)
real_ridge.fit(X_train,y_train)
ridge_intercept = ['constant', real_ridge.intercept_]
ridge_coeftable = pd.DataFrame(columns=["variable", "coefficient"])
ridge_coeftable["variable"]=cols
ridge_coeftable["coefficient"]=real_ridge.coef_
ridge_coeftable = Insert_row_(0, ridge_coeftable, ridge_intercept)
ridge_coeftable
```

```
/opt/conda/lib/python3.8/site-packages/pandas/core/indexing.py:691:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
```

[8]:

	variable	coefficient
0	constant	3.777653
1	fixed acidity	0.019054
2	volatile acidity	-1.032986
3	citric acid	-0.112781
4	residual sugar	0.020685
5	chlorides	-1.206450
6	free sulfur dioxide	0.005368
7	total sulfur dioxide	-0.003462
8	density	0.006622
9	pH	-0.407571
10	sulphates	0.697506
11	alcohol	0.316669

7 2.2 OSR^2, MSE, MAE

[9]:

```python
ridge_osr2=OSR2(real_ridge, X_test, y_test, y_train)

rpredict_train=real_ridge.predict(X_train)
rpredict_test=real_ridge.predict(X_test)

ridge_train_mse=mse(y_train, rpredict_train)
ridge_test_mse=mse(y_test, rpredict_test)

ridge_train_mae=mae(y_train, rpredict_train)
ridge_test_mae=mae(y_test, rpredict_test)

print(ridge_osr2)
print(ridge_train_mse, ridge_test_mse)
print(ridge_train_mae, ridge_test_mae)
```

```
0.3162273169367935
0.40480952854113644 0.45388129853861753
0.49402013323763855 0.5186993050349109
```

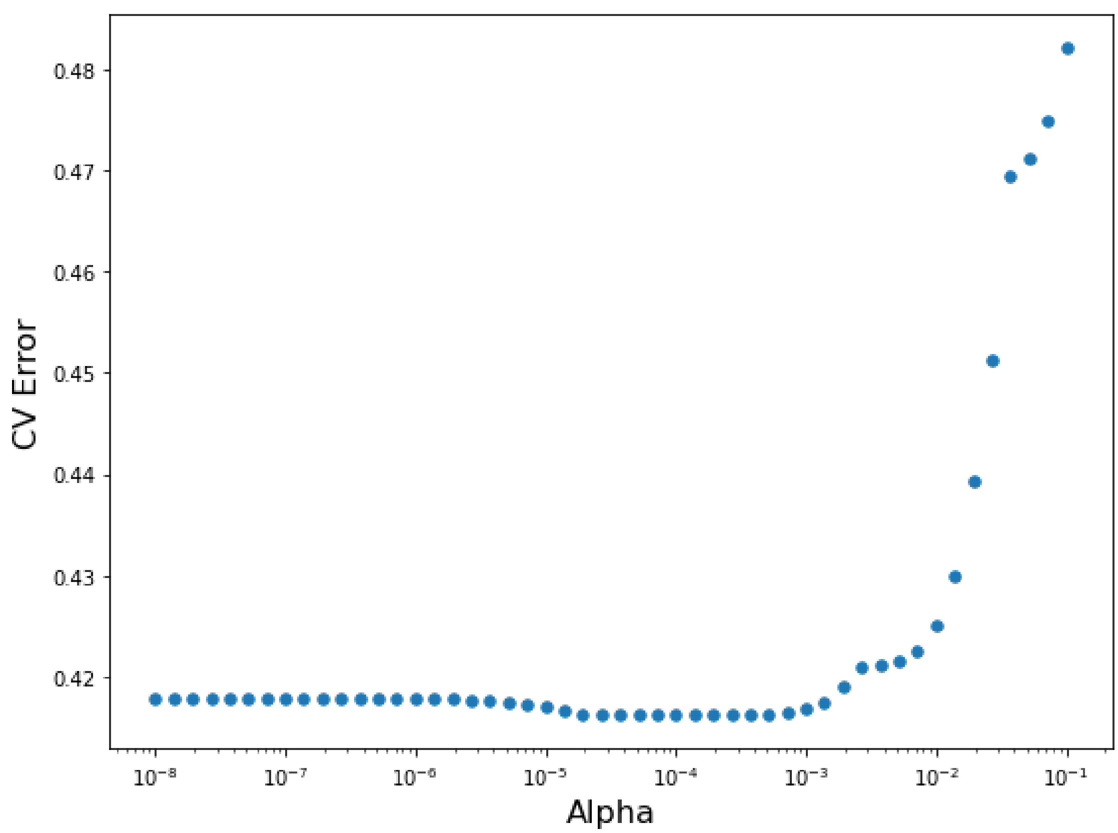
Same as the ridge regression model, I used GridSearchCV with 10 splits, I got a graph of CV error then found a best tuning parameter which results in the lowest CV error. With the best parameter, I made a table of coefficient for the lasso regression model. Also for the lasso model, I assigned an alpha_grid, 88 random states, and a scoring method as a negative mean squared error then fit to X and y. Later in order to find CV error score, I multiplied -1 since I used negative mean squared error. With those, I found a CV error graph of lasso regression(log of Alpha vs CV Error) model. I also found OSR2, MSE, and MAE for the lasso model.

8 3. Lasso Regression CV

[10]:
```
alphas = np.logspace(-8, 1 , num=50, base=10)

for a in alphas:
    lasso = Lasso(alpha=a, random_state=88)

alpha_grid = {'alpha': np.logspace(-8, -1, num=50, base=10)}
lasso_cv = GridSearchCV(lasso, alpha_grid, scoring='neg_mean_squared_error',␣
 ↪cv=10)
lasso_cv.fit(X_train, y_train)
range_alpha = lasso_cv.cv_results_['param_alpha'].data
CV_scores = lasso_cv.cv_results_['mean_test_score']*(-1)
plt.figure(figsize=(8, 6))
ax = plt.gca()
ax.set_xscale('log')
plt.xlabel('Alpha', fontsize=16)
plt.ylabel('CV Error', fontsize=16)
plt.scatter(range_alpha, CV_scores, s=30)
plt.tight_layout()
plt.show()
```



[11]:
```python
print(lasso_cv.best_params_)
```

```
{'alpha': 0.00019306977288832496}
```

With this best parameter, I found a table of coefficients of the lasso regression model below. The coefficients of the lasso regression model (some coefficients are zero) seems to be much closer to 0 than those of OLS and ridge regression models.

9 3.1 Table of Coefficients

[12]:
```python
real_lasso = Lasso(alpha=0.0001, fit_intercept=True)
real_lasso.fit(X_train,y_train)
lasso_intercept=['constant', real_lasso.intercept_]
lasso_coeftable = pd.DataFrame(columns=["variable", "coefficient"])
lasso_coeftable["variable"]=cols
lasso_coeftable["coefficient"]=real_lasso.coef_
lasso_coeftable = Insert_row_(0, lasso_coeftable, lasso_intercept)
lasso_coeftable
```

```
/opt/conda/lib/python3.8/site-packages/pandas/core/indexing.py:691:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value, self.name)
```

[12]:

	variable	coefficient
0	constant	4.103093
1	fixed acidity	0.012627
2	volatile acidity	-1.023197
3	citric acid	-0.081584
4	residual sugar	0.021427
5	chlorides	-1.768867
6	free sulfur dioxide	0.005408
7	total sulfur dioxide	-0.003583
8	density	0.000000
9	pH	-0.470230
10	sulphates	0.765136
11	alcohol	0.310602

10 3.2 OSR^2, MSE, MAE

[13]:
```python
lasso_osr2=OSR2(real_lasso, X_test, y_test, y_train)

lpredict_train=real_lasso.predict(X_train)
```

```
lpredict_test=real_lasso.predict(X_test)

lasso_train_mse=mse(y_train, lpredict_train)
lasso_test_mse=mse(y_test, lpredict_test)

lasso_train_mae=mae(y_train, lpredict_train)
lasso_test_mae=mae(y_test, lpredict_test)

print(lasso_osr2)
print(lasso_train_mse, lasso_test_mse)
print(lasso_train_mae, lasso_test_mae)
```

```
0.3187086407904991
0.4041184304482724 0.45223422119740314
0.49343528411077714 0.5181859261423012
```

Finally, I got a comparison table comparing values of OSR^2, RMSE, and MAE of those three previous models based on the test sets. According to the comparison table, Lasso Regression Model seems to be the most desirable because of the highest OSR^2 accuracy and lowest RMSE and MAE as well. And the ridge regression model seems to be the worst because of the lowest OSR^2 and highest RMSE and MAE.

11 4. Comparison Table

[14]:
```
comparison_data = {
    'Ridge Regression': ['{:.6f}'.format(ridge_osr2),
                         '{:.6f}'.format((ridge_test_mse)**(1/2)),
                         '{:.6f}'.format(ridge_test_mae)],
    'OLS Regression': ['{:.6f}'.format(ols_osr2),
                       '{:.6f}'.format((ols_test_mse)**(1/2)),
                       '{:.6f}'.format(ols_test_mae)],
    'Lasso Regression': ['{:.6f}'.format(lasso_osr2),
                         '{:.6f}'.format((lasso_test_mse)**(1/2)),
                         '{:.6f}'.format(lasso_test_mae)],
}
comparison_table = pd.DataFrame(data=comparison_data, index=['Out-of-sample␣
↪R2', 'Out-of-sample RMSE', 'Out-of-sample MAE'])
comparison_table
```

[14]:

	Ridge Regression	OLS Regression	Lasso Regression
Out-of-sample R2	0.316227	0.315624	0.318709
Out-of-sample RMSE	0.673707	0.674004	0.672484
Out-of-sample MAE	0.518699	0.520095	0.518186

12 5. Predict Wine Quality using Lasso Regression

[21]:
```
lasso_pred0 = lasso_cv.predict(X_train)
lasso_pred1 = lasso_cv.predict(X_test)
lasso_pred0 = lasso_pred0.astype(int)
lasso_pred1 = lasso_pred1.astype(int)
lasso_pred = np.concatenate((lasso_pred0, lasso_pred1))
lasso_pred
```

[21]:
```
array([6, 5, 5, …, 5, 6, 5])
```

[22]:
```
# Make a dataframe and convert from float to int
predicted_dataset = pd.DataFrame()
predicted_dataset['Quality'] = y_test
predicted_dataset['PRED Quality'] = lasso_pred1
predicted_dataset
```

[22]:
```
      Quality  PRED Quality
969         5             5
720         5             5
406         6             5
571         6             6
306         5             5
…         …             …
1411        6             5
1442        5             5
8           7             5
1053        7             6
1176        4             5

[480 rows x 2 columns]
```