Assignment Part-II

Subjective Questions

Question 1

Answer:

What is the optimal value of alpha for ridge and lasso regression? What will be the changes in the model if you choose double the value of alpha for both ridge and lasso? What will be the most important predictor variables after the change is implemented?

```
print(f"The optimal value for Ridge Regression is : {optimalvalue_ridge}")
print(f"The optimal value for Lasso Regression is : {optimalvalue_lasso}")
The optimal value for Ridge Regression is : 0.01
The optimal value for Lasso Regression is : 0.0001
# Doubling Lasso and Ridge Regression's alpha values
optimalvalue_ridge *= 2
optimalvalue_lasso *= 2
print(f"Doubled alpha values of Ridge is {optimalvalue_ridge} and Lasso is {optima
lvalue_lasso}")
Doubled alpha values of Ridge is 0.02 and Lasso is 0.0002
```

Build Lasso Regression model

```
alpha = optimalvalue_lasso
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)
Lasso(alpha=0.0002)
lasso.coef_
array([-0.11816776,  0.40326807,  1.04310063,  0.20154269,  0.01320272,
        0.30796107,  0.3295398 , -0.28630072,  0.43099879,  0.1762002 ,
        0.46486274, -0.0752377 , -0.0096367 , -0.12095382,  0.1175943 ,
        0.1614835 ,  0.07220974,  0.12588766,  0.15207907,  0.11349623,
        0.16800439,  0.1053326 , -0.08579734, -0.01212515,  0.00742725,
       -0.23696381, -0.09273587, -0.1004028 ,  0.05524229, -0.05852677,
       -0.12689143, -0.22122537, -0.10950394,  0.        ,  0.14951854,
        0.06393483])
```

Lasso features and their co-efficients

```
df_lasso = pd.DataFrame(index=X_train.columns)
df_lasso.rows = X_train.columns
df_lasso['Lasso'] = lasso.coef_
df_lasso
```

	Lasso
MSSubClass	-0.118168

	Lasso
LotArea	0.403268
OverallQual	1.043101
OverallCond	0.201543
BsmtUnfSF	0.013203
BsmtFullBath	0.307961
FullBath	0.329540
KitchenAbvGr	-0.286301
TotRmsAbvGrd	0.430999
Fireplaces	0.176200
GarageCars	0.464863
EnclosedPorch	-0.075238
MSZoning_RH	-0.009637
LotShape_IR3	-0.120954

	Lasso
LandContour_HLS	0.117594
LandContour_Low	0.161483
LandContour_Lvl	0.072210
Neighborhood_Crawfor	0.125888
Neighborhood_NoRidge	0.152079
Neighborhood_NridgHt	0.113496
Neighborhood_StoneBr	0.168004
Neighborhood_Veenker	0.105333
BldgType_Twnhs	-0.085797
HouseStyle_2.5Unf	-0.012125
RoofStyle_Mansard	0.007427
Exterior1st_BrkComm	-0.236964
Exterior1st_Stucco	-0.092736

	Lasso
Exterior1st_Wd Sdng	-0.100403
Exterior2nd_Wd Sdng	0.055242
BsmtQual_Fa	-0.058527
HeatingQC_Po	-0.126891
Functional_Maj2	-0.221225
Functional_Sev	-0.109504
SaleType_ConLD	0.000000
SaleType_Oth	0.149519
SaleCondition_Alloca	0.063935

Lasso Regression Model Evaluation

```
y_pred_train = lasso.predict(X_train)
y_pred_test = lasso.predict(X_test)

metric_double_l = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(f"Train r2 score is : {r2_train_lr}")
metric_double_l.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(f"Test r2 score is : {r2_test_lr}")
metric_double_l.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(f"Train RSS score is : {rss1_lr}")
metric_double_l.append(rss1_lr)
```

```
rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(f"Test RSS score is : {rss2_lr}")
metric_double_l.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(f"Train MSE score is : {mse_train_lr}")
metric_double_l.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(f"Test MSE score is : {mse_test_lr}")
metric_double_l.append(mse_test_lr**0.5)
```

```
Train r2 score is : 0.8514527457791814
Test r2 score is : 0.8328649333501988
Train RSS score is : 23.53222787609893
Test RSS score is : 12.418974514787804
Train MSE score is : 0.023025663283854l4
Test MSE score is : 0.028353823093l2284
```

Build Ridge Regression model

```
alpha = optimalvalue_ridge
ridge = Ridge(alpha=alpha)
ridge.fit(X_train, y_train)
```

Out[180]:

```
Ridge(alpha=0.02)
```

```
ridge.coef_
```

```
array([-0.11453502,  0.49769529,  1.02880868,  0.20870043,  0.02323615,
        0.30571474,  0.32641994, -0.32622331,  0.44220508,  0.17049876,
        0.46661689, -0.07664049, -0.01803886, -0.15301168,  0.1355378 ,
        0.17556944,  0.08944529,  0.13511868,  0.15957298,  0.11897933,
        0.18028664,  0.13019827, -0.09222125, -0.02481312,  0.06377628,
       -0.33728585, -0.10829837, -0.11334924,  0.06975914, -0.06184982,
       -0.32566534, -0.27073409, -0.30979916,  0.02772084,  0.36717435,
        0.10859608])
```

Ridge features and their co-efficients

```
df_ridge = pd.DataFrame(index=X_train.columns)
df_ridge.rows = X_train.columns
df_ridge['Ridge'] = ridge.coef_
df_ridge
```

	Ridge
MSSubClass	-0.114535
LotArea	0.497695

	Ridge
OverallQual	1.028809
OverallCond	0.208700
BsmtUnfSF	0.023236
BsmtFullBath	0.305715
FullBath	0.326420
KitchenAbvGr	-0.326223
TotRmsAbvGrd	0.442205
Fireplaces	0.170499
GarageCars	0.466617
EnclosedPorch	-0.076640
MSZoning_RH	-0.018039
LotShape_IR3	-0.153012
LandContour_HLS	0.135538

	Ridge
LandContour_Low	0.175569
LandContour_Lvl	0.089445
Neighborhood_Crawfor	0.135119
Neighborhood_NoRidge	0.159573
Neighborhood_NridgHt	0.118979
Neighborhood_StoneBr	0.180287
Neighborhood_Veenker	0.130198
BldgType_Twnhs	-0.092221
HouseStyle_2.5Unf	-0.024813
RoofStyle_Mansard	0.063776
Exterior1st_BrkComm	-0.337286
Exterior1st_Stucco	-0.108298
Exterior1st_Wd Sdng	-0.113349

	Ridge
Exterior2nd_Wd Sdng	0.069759
BsmtQual_Fa	-0.061850
HeatingQC_Po	-0.325665
Functional_Maj2	-0.270734
Functional_Sev	-0.309799
SaleType_ConLD	0.027721
SaleType_Oth	0.367174
SaleCondition_Alloca	0.108596

Ridge Regression Model Evaluation

```
y_pred_train = ridge.predict(X_train)
y_pred_test = ridge.predict(X_test)

metric_double_r = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(f"Train r2 score is : {r2_train_lr}")
metric_double_r.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(f"Test r2 score is : {r2_test_lr}")
metric_double_r.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(f"Train RSS score is : {rss1_lr}")
metric_double_r.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(f"Test RSS score is : {rss2_lr}")
metric_double_r.append(rss2_lr)
```

```
mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(f"Train MSE score is : {mse_train_lr}")
metric_double_r.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(f"Test MSE score is : {mse_test_lr}")
metric_double_r.append(mse_test_lr**0.5)
```

```
Train r2 score is : 0.8530992328545421
Test r2 score is : 0.8277656774325617
Train RSS score is : 23.271398355851826
Test RSS score is : 12.797874829095916
Train MSE score is : 0.022770448489091807
Test MSE score is : 0.029218892303871955
```

Comparison of co-efficients after Regularization

```
comparison['Ridge_Double'] = ridge.coef_
comparison['Lasso_Double'] = lasso.coef_

comparison.sort_values(by='Lasso', ascending=False)
```

	Linear	Ridge	Lasso	Ridge_Double	Lasso_Double
OverallQual	1.030270	1.029538	1.036706	1.028809	1.043101
GarageCars	0.466611	0.466613	0.465674	0.466617	0.464863
LotArea	0.502954	0.500309	0.453153	0.497695	0.403268
TotRmsAbvGrd	0.442432	0.442320	0.436777	0.442205	0.430999
FullBath	0.325870	0.326146	0.327681	0.326420	0.329540
BsmtFullBath	0.305247	0.305484	0.306589	0.305715	0.307961
SaleType_Oth	0.375355	0.371221	0.262360	0.367174	0.149519

	Linear	Ridge	Lasso	Ridge_Double	Lasso_Double
OverallCond	0.208632	0.208667	0.204988	0.208700	0.201543
Neighborhood_StoneBr	0.180237	0.180262	0.174085	0.180287	0.168004
Fireplaces	0.169985	0.170243	0.173061	0.170499	0.176200
LandContour_Low	0.175476	0.175524	0.168432	0.175569	0.161483
Neighborhood_NoRidge	0.159484	0.159529	0.155763	0.159573	0.152079
Neighborhood_Crawfor	0.135249	0.135184	0.130558	0.135119	0.125888
LandContour_HLS	0.135634	0.135586	0.126574	0.135538	0.117594
Neighborhood_Veenker	0.130385	0.130292	0.117864	0.130198	0.105333
Neighborhood_NridgHt	0.118791	0.118885	0.116130	0.118979	0.113496
SaleCondition_Alloca	0.109234	0.108913	0.086547	0.108596	0.063935
LandContour_Lvl	0.089749	0.089597	0.080940	0.089445	0.072210
Exterior2nd_Wd Sdng	0.070046	0.069902	0.062630	0.069759	0.055242
RoofStyle_Mansard	0.064157	0.063966	0.035776	0.063776	0.007427

	Linear	Ridge	Lasso	Ridge_Double	Lasso_Double
BsmtUnfSF	0.022946	0.023092	0.018083	0.023236	0.013203
SaleType_ConLD	0.027801	0.027761	0.011742	0.027721	0.000000
MSZoning_RH	-0.017834	-0.017937	-0.013784	-0.018039	-0.009637
HouseStyle_2.5Unf	-0.024813	-0.024813	-0.018555	-0.024813	-0.012125
BsmtQual_Fa	-0.061655	-0.061753	-0.060121	-0.061850	-0.058527
EnclosedPorch	-0.076377	-0.076510	-0.075749	-0.076640	-0.075238
BldgType_Twnhs	-0.092248	-0.092235	-0.089108	-0.092221	-0.085797
Exterior1st_Stucco	-0.108609	-0.108453	-0.100692	-0.108298	-0.092736
Exterior1st_Wd Sdng	-0.113638	-0.113493	-0.107008	-0.113349	-0.100403
MSSubClass	-0.114461	-0.114498	-0.116211	-0.114535	-0.118168
LotShape_IR3	-0.154117	-0.153562	-0.137553	-0.153012	-0.120954
Functional_Sev	-0.316220	-0.312977	-0.212884	-0.309799	-0.109504
HeatingQC_Po	-0.332267	-0.328933	-0.229611	-0.325665	-0.126891

	Linear	Ridge	Lasso	Ridge_Double	Lasso_Double
Functional_Maj2	-0.272084	-0.271407	-0.246685	-0.270734	-0.221225
Exterior1st_BrkComm	-0.340590	-0.338929	-0.288809	-0.337286	-0.236964
KitchenAbvGr	-0.327021	-0.326620	-0.306698	-0.326223	-0.286301

Comparison of metrics after Regularization

```
rg_metric = pd.Series(metric_double_r, name = 'Double Ridge Regression')
ls_metric = pd.Series(metric_double_l, name = 'Double Lasso Regression')

final_metric = pd.concat([final_metric, rg_metric, ls_metric], axis = 1)

final_metric
```

	Metric	Linear Regression	Ridge Regression	Lasso Regression	Double Ridge Regression	Double Lasso Regression
0	R2 Score (Train)	0.853101	0.853100	0.852686	0.853099	0.851453
1	R2 Score (Test)	0.827597	0.827683	0.830786	0.827766	0.832865
2	RSS (Train)	23.271138	23.271204	23.336857	23.271398	23.532228
3	RSS (Test)	12.810372	12.804017	12.573482	12.797875	12.418975
4	MSE (Train)	0.150898	0.150898	0.151111	0.150899	0.151742

	Metric	Linear Regression	Ridge Regression	Lasso Regression	Double Ridge Regression	Double Lasso Regression
5	MSE (Test)	0.171019	0.170976	0.169430	0.170935	0.168386

Question 2

You have determined the optimal value of lambda for ridge and lasso regression during the assignment. Now, which one will you choose to apply and why?

```
Answer:

final_metric
```

	Metric	Linear Regression	Ridge Regression	Lasso Regression	Double Ridge Regression	Double Lasso Regression
0	R2 Score (Train)	0.853101	0.853100	0.852686	0.853099	0.851453
1	R2 Score (Test)	0.827597	0.827683	0.830786	0.827766	0.832865
2	RSS (Train)	23.271138	23.271204	23.336857	23.271398	23.532228
3	RSS (Test)	12.810372	12.804017	12.573482	12.797875	12.418975
4	MSE (Train)	0.150898	0.150898	0.151111	0.150899	0.151742
5	MSE (Test)	0.171019	0.170976	0.169430	0.170935	0.168386

Question 3

After building the model, you realised that the five most important predictor variables in the lasso model are not available in the incoming data.
You will now have to create another model excluding the five most important predictor variables. Which are the five most important predictor variables now?

Answer:

Looking at the top 5 important predictor variables in Lasso model

```
comparison.sort_values(by='Lasso',ascending=False).head()
```

	Linear	Ridge	Lasso	Ridge_Double	Lasso_Double
OverallQual	1.030270	1.029538	1.036706	1.028809	1.043101
GarageCars	0.466611	0.466613	0.465674	0.466617	0.464863
LotArea	0.502954	0.500309	0.453153	0.497695	0.403268
TotRmsAbvGrd	0.442432	0.442320	0.436777	0.442205	0.430999
FullBath	0.325870	0.326146	0.327681	0.326420	0.329540

```
# Looking at the current top 5 important predictor variables in Lasso model
comparison.sort_values(by='Lasso',ascending=False).Lasso.head(5)

OverallQual     1.036706
GarageCars      0.465674
LotArea         0.453153
TotRmsAbvGrd    0.436777
FullBath        0.327681
Name: Lasso, dtype: float64

# Creating a list to hold the current top 5 important predictor variables
top5_names = list(comparison['Lasso'].sort_values(ascending=False).head(5).index)
top5_names
 ['OverallQual', 'GarageCars', 'LotArea', 'TotRmsAbvGrd', 'FullBath']
# Drop the top 5 important predictor variables from X_train and X_test
X_train = X_train.drop(top5_names, axis=1)
```

```
X_test = X_test.drop(top5_names, axis=1)

print(X_train.shape)
print(X_test.shape)
(1022, 31)
(438, 31)

# list of alphas to tune

params = { 'alpha' : [0.0001,0.001,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,
                      0.7,0.8,0.9,1.0,2.0,3.0,4.0,5.0,6.0,7.0,
                     8.0,9.0,10.0,20,50,100,500,1000]}

# Applying lasso regression with 5 fold cross validation

lasso = Lasso()
folds = 5
model_cv = GridSearchCV(estimator=lasso,
                        param_grid=params,
                        scoring='neg_mean_absolute_error',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)
model_cv.fit(X_train, y_train)
Fitting 5 folds for each of 28 candidates, totalling 140 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent worker
s.
[Parallel(n_jobs=1)]: Done 140 out of 140 | elapsed:    2.8s finished

GridSearchCV(cv=5, estimator=Lasso(),
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.3,
                                   0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0
,
                                   4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 20, 50,
                                   100, 500, 1000]},
             return_train_score=True, scoring='neg_mean_absolute_error',
             verbose=1)


cv_results.shape
Out[196]:

(28, 21)

# Plotting train scores with alpha

plt.plot(cv_results['param_alpha'], cv_results['mean_train_score'], color='green')
plt.xlabel('alpha')
plt.ylabel("Negative Mean Absolute Error")

plt.title("Neg MAE and Alphas")
plt.show()



# Plotting testing scores with alpha
plt.plot(cv_results['param_alpha'], cv_results['mean_test_score'], color='orange')
plt.xlabel('alpha')
plt.ylabel("Negative Mean Absolute Error")
```

```
plt.title("Neg MAE and Alphas")
plt.show()
```

Getting the optimal value of lambda

```
optimalvalue_lasso = model_cv.best_params_['alpha']
optimalvalue_lasso
```

Out[199]:

```
0.0001
```

Build final Lasso Regression model

```
alpha = optimalvalue_lasso
lasso = Lasso(alpha=alpha)
lasso.fit(X_train, y_train)
```

```
Lasso(alpha=0.0001)
```

```
lasso.coef_
```

```
array([ 0.00228341,  0.21978345,  0.55040169,  0.50982786, -0.32405959,
        0.5890991 , -0.21277777, -0.05180727,  0.08867101,  0.29104398,
        0.18345996,  0.12195927,  0.10987309,  0.48620204,  0.43943659,
        0.42136982,  0.19854878, -0.18712857, -0.        ,  0.0650244 ,
       -0.56986175, -0.07587302, -0.12890651, -0.00653112, -0.15232127,
       -0.28407441, -0.52231084, -0.03715972, -0.        ,  0.00847051,
        0.22868149])
```

Lasso features and their co-efficients

```
df_lasso = pd.DataFrame(index=X_train.columns)
df_lasso.rows = X_train.columns
df_lasso['Lasso'] = lasso.coef_
```

Getting the new top 5 important predictor variables via Lasso Regression

```
df_lasso.sort_values(by='Lasso', ascending=False).head(5)
```

	Lasso
Fireplaces	0.589099

	Lasso
BsmtUnfSF	0.550402
BsmtFullBath	0.509828
Neighborhood_NoRidge	0.486202
Neighborhood_NridgHt	0.439437

Question 4

How can you make sure that a model is robust and generalisable? What are the implications of the same for the accuracy of the model and why?

Answer:

A robust model has low variance. This means that an unprecendented change in one or more features does not significantly alter the value of the predicted variable. Similarly, a generalizable model has reduced model complexity. As the number of features increase in the model, it becomes more complex which usually leads to low bias but high variance. A generalizable model has just enough features that it has as much low variance as possible.

This can be observed from the Bias-Variance tradeoff visual shown below.

The OLS (Ordinary least squares) regression model is very sensitive to outliers and they induce high variance. To reduce this, we can go ahead with regularization (Ridge/Lasso) which include a penalty term in the cost function of the model. This penalty term will move the coefficents of the model towards 0 and thus it reduces model complexity (as feature addition is heavily discouraged). This reduces overfitting in the model.

So regularization gets us high variance with a small trade-off in bias. Thus it helps us build a model which is robust and generalizable. A robust and generalizable model will have a good, consistent train as well as test accuracy.