Advance Regression Assignment– Part II

Question1: 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?

Solution: For Ridge regression the optimal **value is 10** and for Lasso Regression Optimal value is **0.001**

Following is the metric obtained when running with these optimal values.

Sno.	Metric	Ridge Regression	Lasso Regression
0	R2 Score (Train)	**0.926423**	**0.922557**
1	R2 Score (Test)	0.**881347**	0.**885607**
2	RSS (Train)	11.808481	12.428968
3	RSS (Test)	8.551133	8.244143
4	MSE (Train)	0.107543	0.110333
5	MSE (Test)	0.139725	0.137194

Top 10 Predictor variables for Ridge with optimal value of **alpha = 10** are as below.

1. MSZoning_RL 0.092868
2. MSZoning_RM 0.075064
3. OverallQual 0.065915
4. GrLivArea 0.058130
5. MSZoning_FV 0.045185
6. GarageCars 0.043404
7. OverallCond 0.043175
8. 2ndFlrSF 0.040472
9. Neighborhood_NridgHt 0 .037209
10. 1stFlrSF 0.033143

For Lasso regression with **optimal value of alpha=.001** , predictor and coefficients are as follow.

1. GrLivArea 0.107182
2. MSZoning_RL 0.071812
3. OverallQual 0.071131

4. MSZoning_RM 0.050647
5. GarageCars 0.045065
6. OverallCond 0.044631
7. MSZoning_FV 0.032043
8. Neighborhood_NridgHt 0.031903
9. Neighborhood_Somerst 0.028879
10. Neighborhood_Crawfor 0.028291

When we double the alpha for both Ridge and Lasso regression, we obtain the following metrics

New Alpha for **Ridge = 20 and for Lasso =.002 Below is the new Metrics obtained.**

	Metric	Ridge Regression	Lasso Regression
0	R2 Score (Train)	0.925179	0.915170
1	R2 Score (Test)	0.881721	0.884511
2	RSS (Train)	12.008163	13.614575
3	RSS (Test)	8.524197	8.323131
4	MSE (Train)	0.108449	0.115475
5	MSE (Test)	0.139505	0.137850

Changes in Ridge Regression metrics:

- R2 score of train set decreased from 0. **926423** to 0. *925179*
- R2 score of test set almost same 0.881347 → 0.881721

Changes in Lasso metrics:

- R2 score of train set decreased from 0.**922557** to 0.915170
- R2 score of test set decreased from 0.**885607** to 0.884511

Predictor values after double the value of alpha **to 20 and 0.002** for Ridge and Lasso Respectively.

Ridge Regression – top 10 predictor variables after changing alpha values to double (i.e 20)

1. OverallQual 0.065541
2. MSZoning_RL 0.064806
3. GrLivArea 0.055538

4. MSZoning_RM 0.051575
5. OverallCond 0.042988
6. GarageCars 0.040875
7. 2ndFlrSF 0.038110
8. Neighborhood_NridgHt 0.035017
9. 1stFlrSF 0.032313
10. MSZoning_FV 0.031008

Lasso Regression – top 10 parameters after changing alpha values to double (i.e 0.002)

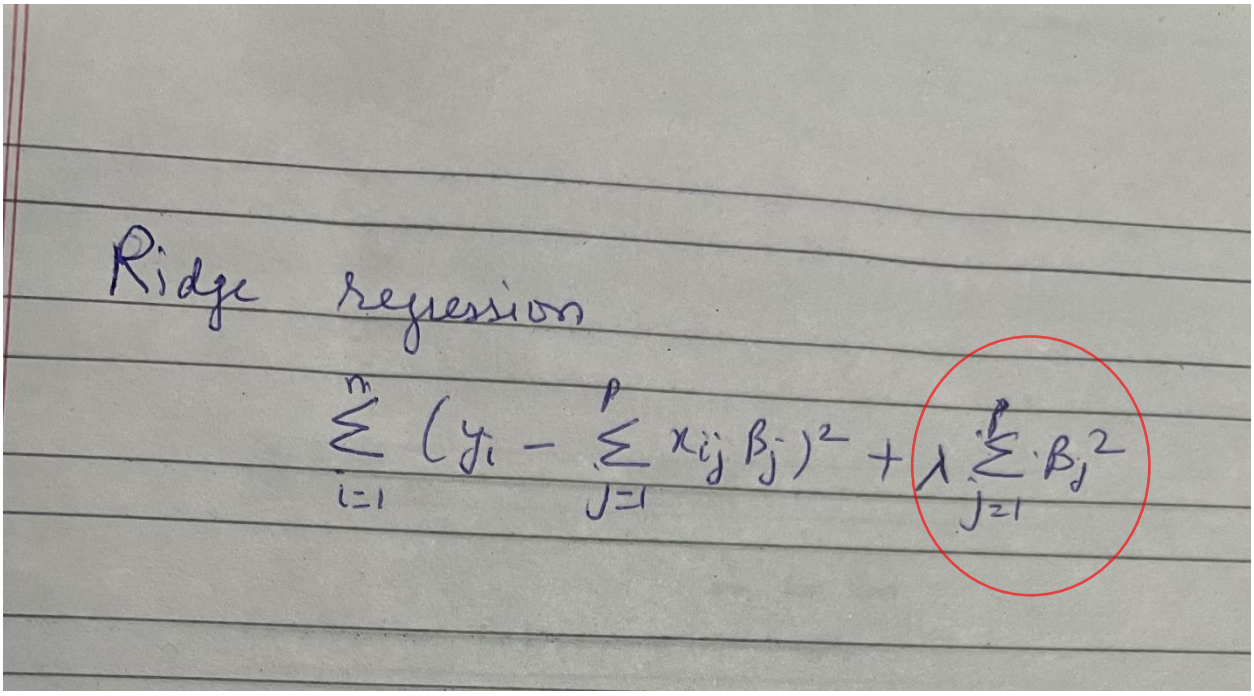
1. GrLivArea 0.099823
2. OverallQual 0.079465
3. OverallCond 0.045405
4. GarageCars 0.043837
5. Neighborhood_NridgHt 0.030023
6. BsmtFullBath 0.028023
7. Neighborhood_Somerst 0.027694
8. Neighborhood_Crawfor 0.026968
9. Condition1_Norm 0.022199
10. SaleType_New 0.019073

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?

Solution: R2(Train) score & R2 (Test) score is comparative in both the regression techniques (Lasso and Ridge). However, if focus is to keep the model simple then Use the **Lasso regression** as it reduces the number of features/variables.

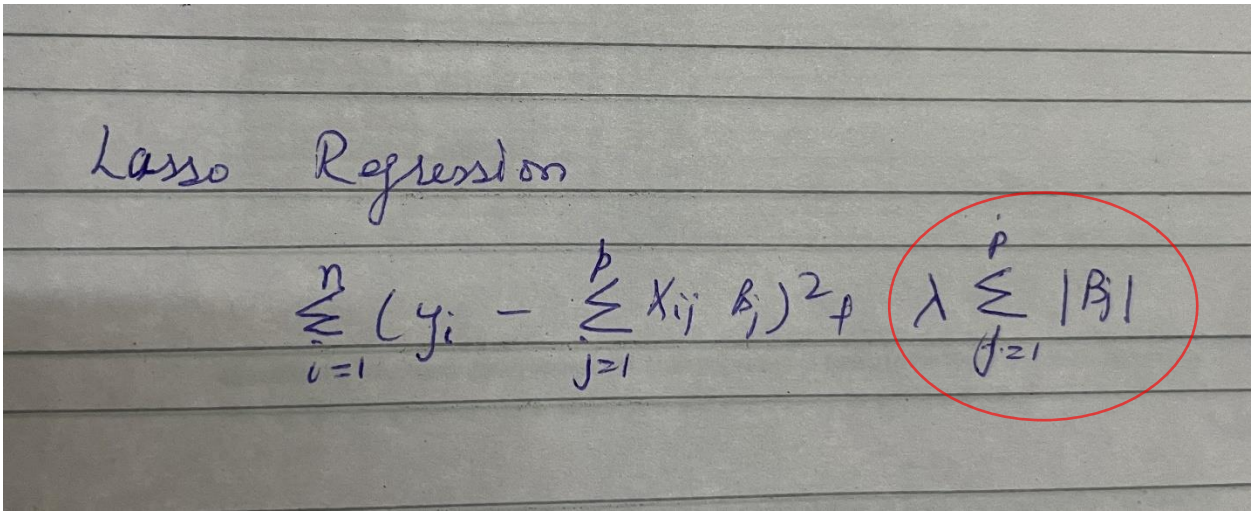
If we need to reduce the coefficient value/weight we can select the **Ridge Regression**.

Ridge regression adds “squared magnitude” of coefficient as penalty term to the loss function. Below is the equation and circled component in equation part represents Ridge regularization element.



Here, if *lambda* is zero then you can imagine we get back OLS. However, if *lambda* is very large then it will add too much weight and it will lead to under-fitting. Having said that it's important how *lambda* is chosen. This technique works very well to avoid over-fitting issue.

Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds "*absolute value of magnitude*" of coefficient as penalty term to the loss function.



The **key difference** between these techniques is that Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for **feature selection** in case we have a huge number of features. Therefore, in this particular case if focus is to eliminate features, then opt for lasso regression.

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?

Solution

After dropping 5 top variables

drop_vars=['MSZoning_RL','MSZoning_RM','OverallQual','GrLivArea','MSZoning_FV']

Drop the top 5 features and rerunning.

In [632]:
```
# Dropping the top 5 predictors and discovering new variables.

drop_vars=['MSZoning_RL','MSZoning_RM','OverallQual','GrLivArea','MSZoning_FV']
```

In [633]:
```
X_train_d= X_train.drop(drop_vars, axis=1)
X_test_d = X_test.drop(drop_vars, axis=1)
```

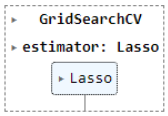
In [651]:
```
lasso = Lasso()

# cross validation
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_d, y_train)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
```

Out[651]:



In [652]:
```
# Printing the best hyperparameter alpha
print(model_cv.best_params_)
```

```
{'alpha': 0.001}
```

In [654]:
```
#Fitting lasso model for alpha = .001 after dropping the 5 top features and printing coefficients which have been penalised
alpha = .001
lasso = Lasso(alpha=alpha)

lasso.fit(X_train_d, y_train)
print(lasso.coef_)
```

```
[-1.89683060e-02 -1.30579051e-02  1.87970676e-02  5.26665913e-02
 -1.06294124e-03 -0.00000000e+00  2.63443721e-03  8.87163081e-03
  8.58489465e-03  9.10413774e-02  9.58024821e-02  5.10379285e-03
  2.52197742e-02  5.18220127e-04  2.05041586e-02  1.29645567e-02
  1.14547070e-02 -1.54082369e-02  1.57545994e-02  0.00000000e+00
 -0.00000000e+00  4.50332153e-02  1.68727035e-03  1.10729394e-02
  0.00000000e+00  7.09874191e-03  7.27686623e-03  1.06161334e-02
 -1.02214875e-02 -4.47215287e-02  3.65118710e-03  3.65559254e-03
```

In [655]:
```
# Lets calculate some metrics such as R2 score, RSS and RMSE
y_pred_train = lasso.predict(X_train_d)
y_pred_test = lasso.predict(X_test_d)

metriclasso = []
r2_train_lr = r2_score(y_train, y_pred_train)
print(r2_train_lr)
metriclasso.append(r2_train_lr)

r2_test_lr = r2_score(y_test, y_pred_test)
print(r2_test_lr)
metriclasso.append(r2_test_lr)

rss1_lr = np.sum(np.square(y_train - y_pred_train))
print(rss1_lr)
metriclasso.append(rss1_lr)

rss2_lr = np.sum(np.square(y_test - y_pred_test))
print(rss2_lr)
metriclasso.append(rss2_lr)

mse_train_lr = mean_squared_error(y_train, y_pred_train)
print(mse_train_lr)
metriclasso.append(mse_train_lr**0.5)

mse_test_lr = mean_squared_error(y_test, y_pred_test)
print(mse_test_lr)
metriclasso.append(mse_test_lr**0.5)
```

```
0.9145038293839214
0.8713072777882814
13.721472130186019
9.274700592952074
0.013439247923786502
0.021175115509023002
```

In [ ]:

In [656]:
```
lr_table = {'Metric': ['R2 Score (Train)','R2 Score (Test)','RSS (Train)','RSS (Test)',
                       'MSE (Train)','MSE (Test)'],
        'Linear Regression': metric
        }

lr_metric = pd.DataFrame(lr_table ,columns = ['Metric'] )

rg_metric = pd.Series(metric5, name = 'Ridge Regression')
ls_metric = pd.Series(metriclasso, name = 'Lasso Regression')
```

In [658]:
```
betas.head()
```

Out[658]:

MSSubClass
LotFrontage
LotArea
OverallCond
MasVnrArea

In [659]:
```
betas['Lasso'] = lasso.coef_
```

In [660]:
```
betas['Lasso'].sort_values(ascending=False)[:10]
```

Out[660]:
```
2ndFlrSF                0.095802
1stFlrSF                0.091041
OverallCond             0.052667
GarageCars              0.045033
Neighborhood_NridgHt    0.039340
Neighborhood_Somerst    0.032138
Neighborhood_Crawfor    0.029608
BsmtFullBath            0.025220
Neighborhood_StoneBr    0.024587
Condition1_Norm         0.021664
Name: Lasso, dtype: float64
```

Below are the new top 5 predicted variables for **Lasso regression** after dropping and re-fitting the model with remaining features/variables.

- 2ndFlrSF 0.095802
- 1stFlrSF 0.091041
- OverallCond 0.052667
- GarageCars 0.045033
- Neighborhood_NridgHt 0.039340

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?

- A model is considered robust when it performs irrespective any variation in the data. Data does not affect its performance much. Simple Model should be selected and should also perform on any new unseen data.
- **Bias Variance Tradeoff** In order to make our model more robust and generalizable, we will have to decrease variance which will lead to some bias. Addition of bias means that accuracy will decrease. Therefore, we need to find some balance between accuracy and complexity.
- To make sure a model is robust and generalizable, we need to ensure it doesn't overfit. This is because an overfitting model has very high variance and a smallest change in data affects the model prediction heavily. Such models will identify all the patterns of a training data but fail to pick up the patterns in unseen test data.

BIAS

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn’t seen before. As a result, such models perform very well on training data but has high error rates on test data.