

Question 1

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?

As per my Gridsearch (hyper turning) to find the best of Alpha it was
Alpha of ridge = 4.0
Alpha of Lasso ==100

So You can see that the Model Evaluation on the r2 square . As we used hyper parameter the R2 square we expect that to be reduced when we double that as the alpha we found is the best . The below output is demonstrated in the Jupyter note book.

You can see that the coefficient mostly remind the same but the Prioority changes little bit .
Example the “TotalBsmSF”took precedence in the Ridge vs Ridge when doubled .

Sine the overall alpha values are small , we don’t expect huge change in the model after doubling the alpha.

```
In [137]: 1 coeficient[['Ridge', 'Ridge - Double']].sort_values(by=['Ridge - Double'],ascending=False).head(20)
```

Out[137]:

| | Ridge | Ridge - Double |
|----------------------|-------------|----------------|
| GrLivArea | 57925.28082 | 48856.72185 |
| 1stFlrSF | 49673.24560 | 42400.14285 |
| OverallQual | 48425.36245 | 41842.23485 |
| BsmtFinSF1 | 38255.78583 | 33046.64796 |
| TotRmsAbvGrd | 35231.45786 | 32443.88648 |
| Neighborhood_StoneBr | 37728.07136 | 31383.39813 |
| TotalBsmSF | 34520.52538 | 31005.16570 |
| 2ndFlrSF | 30596.12783 | 25095.93952 |
| GarageArea | 20871.35590 | 21769.81039 |
| GarageCars | 20463.89902 | 21521.43214 |
| MasVnrArea | 22228.35958 | 21260.10103 |
| FullBath | 19709.88213 | 19637.94225 |
| OverallCond | 26107.86060 | 19317.27190 |
| BsmtExposure_Gd | 19590.53052 | 19274.21616 |
| Neighborhood_NoRidge | 19054.50123 | 18215.32180 |
| Neighborhood_NridgHt | 17289.73095 | 18194.53865 |
| LotArea | 21721.06436 | 17843.71974 |
| Neighborhood_Crawfor | 15546.94435 | 15274.15679 |
| LotFrontage | 15619.74089 | 14369.15795 |
| Fireplaces | 10736.62859 | 13930.54294 |

Out[113]:

| | Metric | Linear Regression | Ridge Regression | Lasso Regression | Ridge Regression Double | Lasso Regression Double |
|---|------------------|--------------------|--------------------|--------------------|-------------------------|-------------------------|
| 0 | R2 Score (Train) | 0.89068 | 0.93521 | 0.93168 | 0.92818 | 0.92298 |
| 1 | R2 Score (Test) | 0.86456 | 0.92320 | 0.91943 | 0.92215 | 0.91401 |
| 2 | RSS (Train) | 553541216983.70300 | 328070961373.04523 | 345960567096.70441 | 363675656679.49542 | 390005939478.66406 |
| 3 | RSS (Test) | 296807333068.61591 | 168308305200.07724 | 176566344688.97595 | 170603672243.01233 | 188431007930.71878 |
| 4 | MSE (Train) | 25549.18529 | 19669.16708 | 20198.32584 | 20709.00294 | 21445.57447 |
| 5 | MSE (Test) | 26647.05218 | 20066.17671 | 20552.55515 | 20202.54332 | 21231.86014 |

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?

The optimal data as per my Modeling is

Ridge – 4.0

Lasso – 100

From the below we can see that the MSE are

Ridge ~ 20066

Lasso ~ 20552

Ridge Regression did better for me as the Test scored and the Training score are good

Generally, Lasso should perform better in situations where only a few among all the predictors that are used to build our model have a significant influence on the response variable. So, feature selection, which removes the unrelated variables, should help. But Ridge should do better when all the variables have almost the same influence on the response variable.

So looks like I will go with the Ridge regression model looking at the R2 Square and also MSE values

Out[113]:

| | Metric | Linear Regression | Ridge Regression | Lasso Regression | Ridge Regression Double | Lasso Regression Double |
|---|------------------|--------------------|--------------------|--------------------|-------------------------|-------------------------|
| 0 | R2 Score (Train) | 0.89068 | 0.93521 | 0.93168 | 0.92818 | 0.92298 |
| 1 | R2 Score (Test) | 0.86456 | 0.92320 | 0.91943 | 0.92215 | 0.91401 |
| 2 | RSS (Train) | 553541216983.70300 | 328070961373.04523 | 345960567096.70441 | 363675656679.49542 | 390005939478.66406 |
| 3 | RSS (Test) | 296807333068.61591 | 168308305200.07724 | 176566344688.97595 | 170603672243.01233 | 188431007930.71878 |
| 4 | MSE (Train) | 25549.18529 | 19669.16708 | 20198.32584 | 20709.00294 | 21445.57447 |
| 5 | MSE (Test) | 26647.05218 | 20066.17671 | 20552.55515 | 20202.54332 | 21231.86014 |

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?

Below are the New top predictors and more you can see from the below

- 1stFlrSF
- 2ndFlrSF
- TotalBsmtSF
- TotRmsAbvGrd
- MasVnrArea
- MasVnrArea

```

R2 Score for the taining set for LASSO : 0.9204005141035227
R2 Score for the Test set for LASSO : 0.9103152223221794
RSS For Train set for LASSO : 403064166453.66144
RSS For Test set for LASSO : 403064166453.66144
MSE for tain set for LASSO : 475311517.0444121
MSE for tain set for LASSO : 470181954.0057159

```

```

In [172]: 1 #important predictor variables
2 coefficeint_after_removal = pd.DataFrame(index=X_train2.columns)
3 coefficeint_after_removal.rows = X_train2.columns
4 coefficeint_after_removal['lasso_remove_top_features'] = lasso_remove_top_features.coef_
5 pd.set_option('display.max_rows', None)
6 coefficeint_after_removal.sort_values(by=['lasso_remove_top_features'],ascending=False)

```

```

Out[172]:

```

| | lasso_remove_top_features |
|----------------------|---------------------------|
| 1stFirSF | 155948.04897 |
| 2ndFirSF | 100456.58411 |
| TotalBsmtSF | 80456.86750 |
| TotRmsAbvGrd | 36662.68378 |
| MasVnrArea | 29670.42352 |
| SaleType_New | 26531.73827 |
| Functional_Typ | 23374.65261 |
| GarageCars | 22666.79925 |
| BsmtExposure_Gd | 20807.69514 |
| GarageArea | 20371.37610 |
| Neighborhood_Crawfor | 19540.94857 |
| YearBuilt | 18339.98192 |
| LotFrontage | 13949.73326 |
| Neighborhood_NoRidge | 13015.83153 |
| Neighborhood_NridgHt | 12790.96276 |
| MSZoning_FV | 10780.89363 |
| MasVnrType_Stone | 9593.12732 |
| GarageType_BuiltIn | 9501.84863 |
| BsmtFinType1_GLQ | 9295.69835 |
| YearRemodAdd | 9125.31120 |
| LandContour_HLS | 8598.73200 |
| SaleCondition_Normal | 8069.22300 |
| MSZoning_RL | 8047.91873 |
| ScreenPorch | 7701.12061 |
| WoodDeckSF | 7700.16017 |
| Condition1_Norm | 7544.08790 |

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?

Occam's razor is perhaps the most important thumb rule in machine learning and is incredibly 'simple' at the same time. When in dilemma, choose the simpler model. The question then is 'how do we define simplicity?'. In the next segment, you will learn about some objective ways to measure model simplicity and understand why simplicity is preferred over sophistication and complexity using various examples.

The model should have more or less the test accuracy as the training score. Outliers shouldn't dominate the data. If the model is not robust then it cannot be considered for predictive analysis

Overfitting

Overfitting is a phenomenon wherein a model becomes highly specific to the data on which it is trained and fails to generalise to other unseen data points in a larger domain. A model that has become highly specific to a training data set has 'learnt' not only the hidden patterns in data but also the noise and the inconsistencies in it. In a typical case of overfitting, a model performs quite well on the training data but fails miserably on the test data.

- A simpler model is usually more generic than a complex model. This becomes important because generic models are bound to perform better on unseen data sets.
- A simpler model requires fewer training data points. This becomes extremely important because in many cases, one has to work with limited data points.
- A simple model is more robust and does not change significantly if the training data points undergo small changes.
- A simple model may make more errors in the training phase but is bound to outperform complex models when it views new data. This happens because of overfitting.

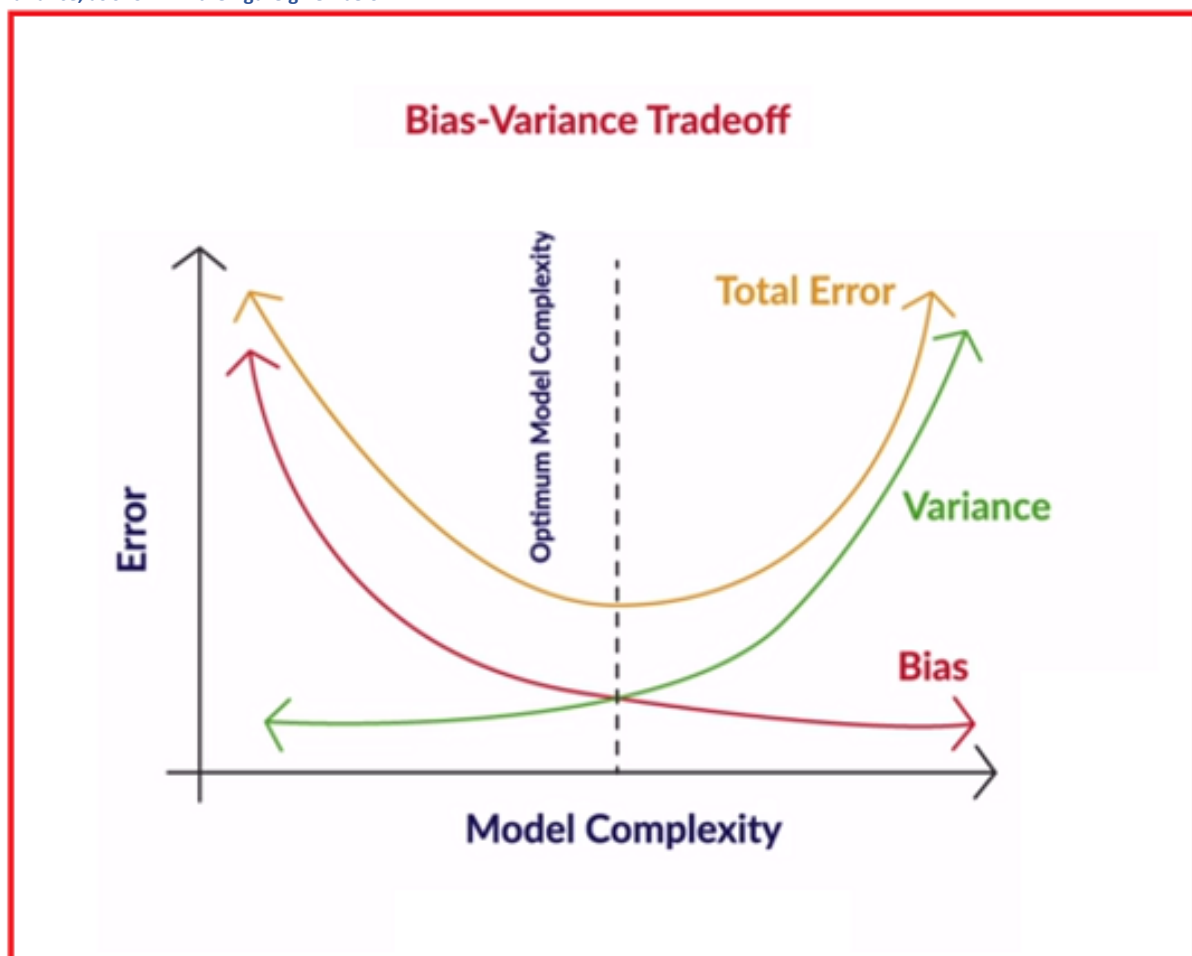
Bias and Variance

We considered the example of a model memorising the entire training data set. If you change the data set slightly, this model will also need to change drastically. The model is, therefore, unstable and sensitive to changes in training data, and this is called high variance.

The 'variance' of a model is the variance in its output on some test data with respect to the changes in the training data. In other words, variance here refers to the degree of changes in the model itself with respect to changes in the training data.

Bias quantifies how accurate the model is likely to be on future (test) data. Extremely simple models are likely to fail in predicting complex real-world phenomena. Simplicity has its own disadvantages.

Ideally, we want to reduce both bias and variance because the expected total error of a model is the sum of the errors in bias and variance, as shown in the figure given below.



Bias Variance Tradeoff

In practice, however, we often cannot have a model with a low bias and a low variance. As the model complexity increases, the bias reduces, whereas the variance increases and, hence, the trade-off.

The output of our calculation from the above it is evident that Total Error = Bias+Variance, we could also see that the MSE calculated from the scikit-library is almost equal to average expected loss.

This is demonstrated in the Jupyter notebook if you want to refer. Do install the mlxtend module .

It can be observed that the Bias has been reduced after regularization and there is a slight increase in variance and the total avg error is also brought down

```
In [199]: 1 from mlxtend.evaluate import bias_variance_decomp
2 from sklearn import metrics
3 #X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
5 ##print(type(X_train))
6 models = [lm_rfe, ridge, lasso]
7
8 for model in models:
9     mse, bias, var = bias_variance_decomp(model, X_train.values, y_train.values, X_test.values, y_test.values, loss
10     y_pred=model.predict(X_test)
11     # summarize results
12     print()
13     print('MSE for from bias_variance lib [avg expected loss]: %.3f' % mse)
14     print('Avg for Bias : %.3f' % bias)
15     print('Avg for Variance : %.3f' % var)
16     print('Mean for Square error by Skit-learn lib : %.3f' % metrics.mean_squared_error(y_test,y_pred))
17
```


| | |
|---|-----------------|
| MSE for from bias_variance lib [avg expected loss]: | 741227611.708 |
| Avg for Bias | : 511918691.077 |
| Avg for Variance | : 229308920.630 |
| Mean for Square error by Skit-learn lib | : 987085612.466 |
| | |
| MSE for from bias_variance lib [avg expected loss]: | 471148472.931 |
| Avg for Bias | : 400011499.972 |
| Avg for Variance | : 71136972.960 |
| Mean for Square error by Skit-learn lib | : 567372034.120 |
| | |
| MSE for from bias_variance lib [avg expected loss]: | 464035996.410 |
| Avg for Bias | : 404288075.437 |
| Avg for Variance | : 59747920.973 |
| Mean for Square error by Skit-learn lib | : 534948830.194 |

In []: 1