Surprise Housing Case Study Assignment2

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 to double the value of alpha for both ridge and lasso? What will be the most important predictor variables after the change is implemented?

Answer1

Optimal value of Alpha:

For Ridge = 2

For Lasso = 0.0001

If we double the value of alpha i.e.,

For Ridge, it will become 4 and

For Lasso, it will become 0.0002.

```
# Let us build the ridge regression model with double value of alpha i.e. 4
ridge = Ridge(alpha=4)

# Fit the model on training data
ridge.fit(X_train_rfe, y_train)

# Make predictions
y_train_pred = ridge.predict(X_train_rfe)
y_pred = ridge.predict(X_test_rfe)

# Evaluation
model_eval()
```

```
R-Square for train:0.9402386462104351
R-Square for test:0.8700172391196719
RMSE for train:0.09237954826075229
RMSE for test:0.1378169687693749
```

```python
# Let us build the Lasso regression model with double value of alpha i.e. .0002
lasso = Lasso(alpha=0.0002)

# Fit the model on training data
lasso.fit(X_train_rfe, y_train)

# Make predictions
y_train_pred = ridge.predict(X_train_rfe)
y_pred = ridge.predict(X_test_rfe)

# Evaluation
model_eval()
```

```
R-Square for train:0.9402386462104351
R-Square for test:0.8700172391196719
RMSE for train:0.09237954826075229
RMSE for test:0.1378169687693749
```

Ridge for alpha = 2.0 :

R-Square for train:0.9436190965526192
R-Square for test:0.8659669311749246
RMSE for train:0.08972875378486447
RMSE for test:0.13994770973204126

Ridge for alpha = 4.0 :

R-Square for train:0.9402386462104351
R-Square for test:0.8659669311749246
RMSE for train:0.09237954826075229
RMSE for test:0.13994770973204126

Here:
R2 score for train is decreased from .943 to .940
R2 score for test is almost same
RMSE for train is increased from .08 to .09
RMSE for test is same.

Lasso for alpha 0.0001
R-Square for train:0.945342320671325
R-Square for test:0.8700172391196719
RMSE for train:0.088346879586147
RMSE for test:0.1378169687693749

Lasso for alpha 0.0002
R-Square for train:0.9402386462104351
R-Square for test:0.8659669311749246
RMSE for train:0.09237954826075229
RMSE for test:0.13994770973204126

Here:
R2 score of train is decreased from .945 to 0.940
R2 score of test is also decreased from .87 to .86
RMSE for train is increased from .08 to .09
RMSE for test is also increased from .137 to .139

So, the most important predictor variables after we double the alpha values are:-

- MSZoning_RL
- MSZoning_FV
- MSZoning_RH
- MSZoning_RM
- OverallQual_9
- OverallQual_8
- GrLivArea
- Exterior1st_BrkFace
- CentralAir_Y
- Functional_Typ
- OverallQual_7

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?

Answer2

- The model we will choose to apply will depend on the use case.
- If we have too many variables and one of our primary goal is feature selection, then we will use Lasso Regression.
- If we don't want to get too large coefficients and reduction of coefficient magnitude is one of our prime goals, then we will use Ridge Regression.
- If we take into consideration the value of R2 score and RMSE then based on the value achieved, Lasso seems better with R2 score of 0.86 and RMSE of .137 which is slightly better than Ridge in our case

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 3

Top 5 predictors for Lasso are:

- MSZoning_FV 1.417651
- MSZoning_RH 1.411549
- MSZoning_RL 1.408965
- MSZoning_RM 1.357499
- OverallQual_9 1.241290

After dropping the top 5 features and building model, we get:

```
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]}

# Taking instance of Lasso regressor
lasso = Lasso()

# Calling function for model fit
Lasso_model,Lasso_alpha = reg_model(params,X_train_rfe,y_train,lasso)

# getting the instance on best alpha value
Lasso_fin_model = Lasso(alpha = Lasso_alpha)

# fitting the model on training data set
Lasso_fin_model.fit(X_train_rfe,y_train)

# getting predicted values for train and test dataset
y_train_pred = Lasso_fin_model.predict(X_train_rfe)
y_test_pred = Lasso_fin_model.predict(X_test_rfe)

# calling the model evaluation function
model_eval()

lasso_cv_result = pd.DataFrame(Lasso_model.cv_results_)
```

```
Fitting 5 folds for each of 28 candidates, totalling 140 fits
Optimal value of Alpha is :{'alpha': 0.0001}
R-Square for train:0.9385554628458458
R-Square for test:0.8606413074661882
RMSE for train:0.09367145349787905
RMSE for test:0.1427009441228698
```

Top 5 predictors now are:

```
# 5 top significant LAsso Coefficients
lasso_coeffs = np.exp(Coefficients['Lasso'])
lasso_coeffs.sort_values(ascending=False)[:5]
```

```
GrLivArea              1.138239
Exterior1st_BrkFace    1.111491
CentralAir_Y           1.110912
Functional_Typ         1.109686
GarageYrBlt_2008.0     1.103133
Name: Lasso, dtype: float64
```

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 4

Ensuring that a model is robust and generalizable is crucial for its effectiveness and reliability in real-world applications. Here are some key considerations to achieve robustness and generalizability:

1. **High-quality and diverse training data**: Start with a large and diverse dataset that accurately represents the problem you're trying to solve. The dataset should cover a wide range of scenarios, including edge cases and potential sources of variation. This helps the model learn to generalize patterns and make accurate predictions across different inputs.
2. **Data pre-processing and normalization**: Clean and pre-process the data effectively to remove noise, handle missing values, and normalize the features. Data normalization techniques such as scaling, or standardization can help make the model more robust to variations in feature scales.
3. **Feature engineering**: Carefully select and engineer relevant features that capture the underlying patterns in the data. Domain knowledge and understanding the problem at hand are crucial for effective feature engineering. A well-designed set of features can enhance the model's ability to generalize and improve its robustness.
4. **Regularization techniques**: Apply regularization techniques like L1 or L2 regularization, dropout, or early stopping to prevent overfitting. Overfitting occurs when a model becomes too specific to the training data and fails to generalize well to new, unseen data. Regularization helps to control model complexity and encourages the learning of more generalizable patterns.
5. **Cross-validation and evaluation:** Use techniques like cross-validation to assess the model's performance on unseen data. Cross-validation involves splitting the dataset into multiple subsets, training the model on a portion of the data, and evaluating it on the remaining data. This approach provides a more comprehensive understanding of how the model performs across different data samples and helps detect overfitting.

6. **Hyperparameter tuning:** Optimize the model's hyperparameters using techniques like grid search, random search, or Bayesian optimization. Hyperparameters control the behaviour and performance of the model, and finding the right combination can improve its generalization ability.
7. **Testing on independent datasets**: Validate the model's performance on independent test datasets that were not used during training or hyperparameter tuning. This helps assess the model's ability to generalize to new, unseen data and provides a more accurate measure of its real-world performance.

Implications for model accuracy:

While ensuring robustness and generalizability is crucial, it's important to note that there can be a trade-off between accuracy and generalization. A model that is too complex or overfit to the training data may achieve high accuracy on the training set but fail to generalize well to new data, resulting in lower accuracy on unseen data. To strike the right balance, it's important to monitor the model's performance on both the training and test/validation datasets. If the model shows a significant drop in performance on the test/validation set compared to the training set, it may indicate overfitting. In such cases, regularization techniques and hyperparameter tuning can be applied to improve both accuracy and generalization.

By focusing on robustness and generalizability, you aim to build a model that performs well not just on the training data but also on real-world data, leading to more reliable and accurate predictions in practical applications.