index

March 2, 2022

1 Ridge and Lasso Regression - Lab

1.1 Introduction

In this lab, you'll practice your knowledge of Ridge and Lasso regression!

1.2 Objectives

In this lab you will:

- Use Lasso and Ridge regression with scikit-learn
- Compare and contrast Lasso, Ridge and non-regularized regression

1.3 Housing Prices Data

Let's look at yet another house pricing dataset:

[106]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('Housing_Prices/train.csv')
```

Look at `.info()` of the data:

[107]:
```python
# Your code here
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
```

5	Street	1460 non-null	object
6	Alley	91 non-null	object
7	LotShape	1460 non-null	object
8	LandContour	1460 non-null	object
9	Utilities	1460 non-null	object
10	LotConfig	1460 non-null	object
11	LandSlope	1460 non-null	object
12	Neighborhood	1460 non-null	object
13	Condition1	1460 non-null	object
14	Condition2	1460 non-null	object
15	BldgType	1460 non-null	object
16	HouseStyle	1460 non-null	object
17	OverallQual	1460 non-null	int64
18	OverallCond	1460 non-null	int64
19	YearBuilt	1460 non-null	int64
20	YearRemodAdd	1460 non-null	int64
21	RoofStyle	1460 non-null	object
22	RoofMatl	1460 non-null	object
23	Exterior1st	1460 non-null	object
24	Exterior2nd	1460 non-null	object
25	MasVnrType	1452 non-null	object
26	MasVnrArea	1452 non-null	float64
27	ExterQual	1460 non-null	object
28	ExterCond	1460 non-null	object
29	Foundation	1460 non-null	object
30	BsmtQual	1423 non-null	object
31	BsmtCond	1423 non-null	object
32	BsmtExposure	1422 non-null	object
33	BsmtFinType1	1423 non-null	object
34	BsmtFinSF1	1460 non-null	int64
35	BsmtFinType2	1422 non-null	object
36	BsmtFinSF2	1460 non-null	int64
37	BsmtUnfSF	1460 non-null	int64
38	TotalBsmtSF	1460 non-null	int64
39	Heating	1460 non-null	object
40	HeatingQC	1460 non-null	object
41	CentralAir	1460 non-null	object
42	Electrical	1459 non-null	object
43	1stFlrSF	1460 non-null	int64
44	2ndFlrSF	1460 non-null	int64
45	LowQualFinSF	1460 non-null	int64
46	GrLivArea	1460 non-null	int64
47	BsmtFullBath	1460 non-null	int64
48	BsmtHalfBath	1460 non-null	int64
49	FullBath	1460 non-null	int64
50	HalfBath	1460 non-null	int64
51	BedroomAbvGr	1460 non-null	int64
52	KitchenAbvGr	1460 non-null	int64

```
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

- First, split the data into `X` (predictor) and `y` (target) variables
- Split the data into 75-25 training-test sets. Set the `random_state` to 10
- Remove all columns of `object` type from `X_train` and `X_test` and assign them to `X_train_cont` and `X_test_cont`, respectively

[109]:
```
# Create X and y
y = df["SalePrice"]
X = df.drop(columns = ["SalePrice"], axis = 1)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state = 10)

# Remove "object"-type features from X
cont_features = X.select_dtypes(include=['float64', "int64"])

# Remove "object"-type features from X_train and X_test
X_train_cont = X_train.select_dtypes(include=['float64', "int64"])
```

```
X_test_cont = X_test.select_dtypes(include=['float64', "int64"])
```

1.4 Let's use this data to build a first naive linear regression model

- Fill the missing values in data using median of the columns (use `SimpleImputer`)
- Fit a linear regression model to this data
- Compute the R-squared and the MSE for both the training and test sets

[110]:
```
from sklearn.metrics import mean_squared_error, mean_squared_log_error
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score

# Impute missing values with median using SimpleImputer
impute = SimpleImputer(strategy='median')
X_train_imputed = impute.fit_transform(X_train_cont)
X_test_imputed = impute.transform(X_test_cont)

# Fit the model and print R2 and MSE for training and test sets
linreg = LinearRegression()
linreg.fit(X_train_imputed, y_train)

# Print R2 and MSE for training and test sets
print("Train Set r2 : ",r2_score(y_train, linreg.predict(X_train_imputed)))
print("Test  Set r2 : ",r2_score(y_test, linreg.predict(X_test_imputed)))
print("Mean Squared Error Train: ",
      mean_squared_error(y_train, linreg.predict(X_train_imputed),␣
 ↪squared=True))
print("Mean Squared Error Test: ",
      mean_squared_error(y_test, linreg.predict(X_test_imputed), squared=True))

### From GitHub
print()
print()
print("From GitHub")
print()
print('Training r^2:', linreg.score(X_train_imputed, y_train))
print('Test r^2:', linreg.score(X_test_imputed, y_test))
print('Training MSE:', mean_squared_error(y_train, linreg.
 ↪predict(X_train_imputed)))
print('Test MSE:', mean_squared_error(y_test, linreg.predict(X_test_imputed)))
```

```
Train Set r2 :  0.8069714678400264
Test  Set r2 :  0.8203264293698825
Mean Squared Error Train:  1212415985.7084072
Mean Squared Error Test:  1146350639.8806374
```

```
From GitHub

Training r^2: 0.8069714678400264
Test r^2: 0.8203264293698825
Training MSE: 1212415985.7084072
Test MSE: 1146350639.8806374
```

1.5 Normalize your data

- Normalize your data using a `StandardScalar`

- Fit a linear regression model to this data
- Compute the R-squared and the MSE for both the training and test sets

[112]:
```
from sklearn.preprocessing import StandardScaler

# Scale the train and test data
ss = StandardScaler()
X_train_imputed_scaled = ss.fit_transform(X_train_imputed)
X_test_imputed_scaled = ss.transform(X_test_imputed)

# Fit the model
linreg_norm = LinearRegression()
linreg_norm.fit(X_train_imputed_scaled, y_train)

y_train_predict = linreg_norm.predict(X_train_imputed_scaled)
y_test_predict  = linreg_norm.predict(X_test_imputed_scaled)


# Print R2 and MSE for training and test sets
print("Scaled Train Set r2: " , r2_score(y_train, y_train_predict))
print("Scaled Test Set r2 : " , r2_score(y_test, y_test_predict))
print("Scaled Train Set Mean Squared Error: ",
      mean_squared_error(y_train, y_train_predict, squared = True))
print("Scaled Test Set Mean Squared Error: ",
      mean_squared_error(y_test, y_test_predict, squared = True))

## From GitHub
print()
print("From GitHub")
print('Training r^2:', linreg_norm.score(X_train_imputed_scaled, y_train))
print('Test r^2:', linreg_norm.score(X_test_imputed_scaled, y_test))
print('Training MSE:', mean_squared_error(y_train, linreg_norm.
 ↪predict(X_train_imputed_scaled)))
print('Test MSE:', mean_squared_error(y_test, linreg_norm.
 ↪predict(X_test_imputed_scaled)))
```

```
Scaled Train Set r2:  0.8069732144369715
Scaled Test Set r2 :  0.8203389046729641
Scaled Train Set Mean Squared Error:  1212405015.2988358
Scaled Test Set Mean Squared Error:  1146271045.1376815

From GitHub
Training r^2: 0.8069732144369715
Test r^2: 0.8203389046729641
Training MSE: 1212405015.2988358
Test MSE: 1146271045.1376815
```

1.6 Include categorical variables

The above models didn't include categorical variables so far, let's include them!

- Include all columns of `object` type from `X_train` and `X_test` and assign them to `X_train_cat` and `X_test_cat`, respectively
- Fill missing values in all these columns with the string `'missing'`

[113]:
```
# Create X_cat which contains only the categorical variables
features_cat = X.select_dtypes(include=['object'])

X_train_cat = X_train.select_dtypes(include=['object'])
X_test_cat  = X_test.select_dtypes(include=['object'])

# Fill missing values with the string 'missing'
X_train_cat.fillna(value = "missing", inplace = True)
X_test_cat.fillna(value  = "missing", inplace = True)
```

- One-hot encode all these categorical columns using `OneHotEncoder`
- Transform the training and test DataFrames (`X_train_cat`) and (`X_test_cat`)
- Run the given code to convert these transformed features into DataFrames

[114]:
```
from sklearn.preprocessing import OneHotEncoder

# OneHotEncode categorical variables
ohe = OneHotEncoder(handle_unknown='ignore')

# Transform training and test sets
X_train_ohe = ohe.fit_transform(X_train_cat)
X_test_ohe = ohe.transform(X_test_cat)

# Convert these columns into a DataFrame
columns = ohe.get_feature_names(input_features=X_train_cat.columns)
cat_train_df = pd.DataFrame(X_train_ohe.todense(), columns=columns)
cat_test_df = pd.DataFrame(X_test_ohe.todense(), columns=columns)
```

- Combine `X_train_imputed_scaled` and `cat_train_df` into a single DataFrame

- Similarly, combine `X_test_imputed_scaled` and `cat_test_df` into a single DataFrame

[115]:
```
# Your code here

## My Work
X_train_all = pd.concat([pd.DataFrame(X_train_ohe.todense()),
                         pd.DataFrame(X_train_imputed_scaled)], axis = 1)
X_test_all = pd.concat([pd.DataFrame(X_test_ohe.todense()),
                         pd.DataFrame(X_test_imputed_scaled)], axis = 1)
```

Now build a linear regression model using all the features (`X_train_all`). Also, print the R-squared and the MSE for both the training and test sets.

[117]:
```
# Your code here
from sklearn.linear_model import LinearRegression

linear_model = LinearRegression()
linear_model.fit(X_train_all, y_train)

y_train_predict_lm = linear_model.predict(X_train_all)
y_test_predict_lm  = linear_model.predict(X_test_all)

print("Train Set r2: ", r2_score(y_train_predict_lm, y_train))
print("Test Set r2 : ", r2_score(y_test_predict_lm , y_test))
print("Train MSE   : ", mean_squared_error(y_train, y_train_predict_lm,
                                           squared = True))
print("Train MSE   : ", mean_squared_error(y_test, y_test_predict_lm,
                                           squared = True))

## From GitHub
print()
print("From GitHub")
print('Training r^2:', linear_model.score(X_train_all, y_train))
print('Test r^2:', linear_model.score(X_test_all, y_test))
print('Training MSE:', mean_squared_error(y_train, linear_model.
 ↪predict(X_train_all)))
print('Test MSE:', mean_squared_error(y_test, linear_model.predict(X_test_all)))
```

```
Train Set r2:  0.9316575480222399
Test Set r2 :  -0.00020727560854050253
Train MSE   :  402561357.9616438
Train MSE   :  3.9077733493671256e+30

From GitHub
Training r^2: 0.9359082782249372
Test r^2: -6.12485889105547e+20
Training MSE: 402561357.9616438
Test MSE: 3.9077733493671256e+30
```

Notice the severe overfitting above; our training R-squared is very high, but the test R-squared is negative! Similarly, the scale of the test MSE is orders of magnitude higher than that of the training MSE.

1.7 Ridge and Lasso regression

Use all the data (normalized features and dummy categorical variables, `X_train_all`) to build two models - one each for Lasso and Ridge regression. Each time, look at R-squared and MSE.

1.8 Lasso

With default parameter (alpha = 1)

[119]:
```python
# Your code here
from sklearn.linear_model import Ridge, Lasso, LinearRegression

lasso = Lasso(alpha = 1)
lasso.fit(X_train_all, y_train)

y_train_lasso = lasso.predict(X_train_all)
y_test_lasso  = lasso.predict(X_test_all)

print("Train Set r2: ", r2_score(y_train_lasso, y_train))
print("Test Set r2 : ", r2_score(y_test_lasso , y_test))
print("Train MSE   : ", mean_squared_error(y_train, y_train_lasso,
                                          squared = True))
print("Train MSE   : ", mean_squared_error(y_test, y_test_lasso,
                                          squared = True))



#### FRom GitHub
print()
print("From GitHub")
lasso = Lasso() # Lasso is also known as the L1 norm
lasso.fit(X_train_all, y_train)

print('Training r^2:', lasso.score(X_train_all, y_train))
print('Test r^2:', lasso.score(X_test_all, y_test))
print('Training MSE:', mean_squared_error(y_train, lasso.predict(X_train_all)))
print('Test MSE:', mean_squared_error(y_test, lasso.predict(X_test_all)))
```

```
Train Set r2:  0.9315029672753936
Test Set r2 :  0.8835382251539015
Train MSE   :  402187309.3248636
Train MSE   :  709924619.1651285

From GitHub
Training r^2: 0.9359678304414744
```

```
Test r^2: 0.8887297771152233
Training MSE: 402187309.3248636
Test MSE: 709924619.1651285
```

With a higher regularization parameter (alpha = 10)

[120]:
```python
# Your code here
from sklearn.linear_model import Ridge, Lasso, LinearRegression
from sklearn.metrics import r2_score

lasso_10 = Lasso(alpha = 10)
lasso_10.fit(X_train_all, y_train)

y_train_lasso_10 = lasso_10.predict(X_train_all)
y_test_lasso_10  = lasso_10.predict(X_test_all)

print("Train Set r2: ", r2_score(y_train_lasso_10, y_train))
print("Test Set r2 : ", r2_score(y_test_lasso_10, y_test))
print("Train MSE   : ", mean_squared_error(y_train, y_train_lasso_10,
                                           squared = True))
print("Train MSE   : ", mean_squared_error(y_test, y_test_lasso_10,
                                           squared = True))



#### FRom GitHub
print()
print("From GitHub")
lasso = Lasso() # Lasso is also known as the L1 norm
lasso.fit(X_train_all, y_train)

print('Training r^2:', lasso_10.score(X_train_all, y_train))
print('Test r^2:', lasso_10.score(X_test_all, y_test))
print('Training MSE:', mean_squared_error(y_train, lasso_10.
 ↪predict(X_train_all)))
print('Test MSE:', mean_squared_error(y_test, lasso_10.predict(X_test_all)))
```

```
Train Set r2:  0.9291685420643658
Test Set r2 :  0.8909554463885028
Train MSE   :  412143669.00169474
Train MSE   :  659301974.1202035

From GitHub
Training r^2: 0.9343826801987114
Test r^2: 0.8966641307706718
Training MSE: 412143669.00169474
Test MSE: 659301974.1202035
```

1.9 Ridge

With default parameter (alpha = 1)

[128]:
```
# Your code here
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error, mean_squared_log_error

ridge = Ridge(alpha = 1)
ridge.fit(X_train_all, y_train)
y_train_ridge = ridge.predict(X_train_all)
y_test_ridge = ridge.predict(X_test_all)

print("Train r2  : ", r2_score(y_train, y_train_ridge))
print("Test r2   : ", r2_score(y_test, y_test_ridge))
print("Train MSE : ", mean_squared_error(y_train,y_train_ridge, squared = True))
print("Test MSE  : ", mean_squared_error(y_test,y_test_ridge, squared = True))

### From GitHub

print()
print("From GitHub")
print('Training r^2:', ridge.score(X_train_all, y_train))
print('Test r^2:', ridge.score(X_test_all, y_test))
print('Training MSE:', mean_squared_error(y_train, ridge.predict(X_train_all)))
print('Test MSE:', mean_squared_error(y_test, ridge.predict(X_test_all)))
```

```
Train r2  :  0.9231940244796031
Test r2   :  0.8842330485444209
Train MSE :  482419834.39879984
Test MSE  :  738614579.8334165

From GitHub
Training r^2: 0.9231940244796031
Test r^2: 0.8842330485444209
Training MSE: 482419834.39879984
Test MSE: 738614579.8334165
```

With default parameter (alpha = 10)

[133]:
```
# Your code here
from sklearn.linear_model import Ridge
from sklearn.metrics import r2_score, mean_squared_error

ridge_10 = Ridge(alpha = 10)
ridge_10.fit(X_train_all,y_train)
y_train_ridge_10 = ridge_10.predict(X_train_all)
y_test_ridge_10 = ridge_10.predict(X_test_all)
```

```
print("Train r2  : ", r2_score(y_train, y_train_ridge_10))
print("Test r2   : ", r2_score(y_test, y_test_ridge_10))
print("Train MSE : ", mean_squared_error(y_train,y_train_ridge_10
                                      , squared = True))
print("Test MSE  : ", mean_squared_error(y_test,y_test_ridge_10
                                      , squared = True))

### From GitHub

print()
print("From GitHub")
print('Training r^2:', ridge_10.score(X_train_all, y_train))
print('Test r^2:', ridge_10.score(X_test_all, y_test))
print('Training MSE:', mean_squared_error(y_train, ridge_10.
 ↪predict(X_train_all)))
print('Test MSE:', mean_squared_error(y_test, ridge_10.predict(X_test_all)))
```

```
Train r2  :  0.8990002650425939
Test r2   :  0.8834542222982165
Train MSE :  634381310.5991354
Test MSE  :  743583635.4522319

From GitHub
Training r^2: 0.8990002650425939
Test r^2: 0.8834542222982165
Training MSE: 634381310.5991354
Test MSE: 743583635.4522319
```

1.10 Compare the metrics

Write your conclusions here: ____

1.11 Compare number of parameter estimates that are (very close to) 0 for Ridge and Lasso

Use 10**(-10) as an estimate that is very close to 0.

[136]:
```
# Number of Ridge params almost zero
print("Ridge, alpha = 1")
print(len(ridge.coef_))
print(sum(abs(ridge.coef_) < 10**(-10))/ len(ridge.coef_))

print()
print("Ridge, alpha = 10")
print(len(ridge_10.coef_))
print(sum(abs(ridge_10.coef_) < 10**(-10))/ len(ridge_10.coef_))
```

```
Ridge, alpha = 1
296
0.0

Ridge, alpha = 10
296
0.0
```

[ ]:
```
# Number of Lasso params almost zero
```

[137]:
```
print("Lasso, alpha = 1")
print(len(lasso.coef_))
print(sum(abs(lasso.coef_) < 10**(-10))/ len(lasso.coef_))

print()
print("Lasso, alpha = 10")
print(len(lasso_10.coef_))
print(sum(abs(lasso_10.coef_) < 10**(-10))/ len(lasso_10.coef_))
```

```
Lasso, alpha = 1
296
0.12837837837837837

Lasso, alpha = 10
296
0.26013513513513514
```

Lasso was very effective to essentially perform variable selection and remove about 25% of the variables from your model!

1.12 Put it all together

To bring all of our work together lets take a moment to put all of our preprocessing steps for categorical and continuous variables into one function. This function should take in our features as a dataframe `X` and target as a Series `y` and return a training and test DataFrames with all of our preprocessed features along with training and test targets.

[190]:
```
def preprocess(X, y):
    '''Takes in features and target and implements all preprocessing steps
    for categorical and continuous features returning train and test
    DataFrames with targets'''

    from sklearn.metrics import mean_squared_error, mean_squared_log_error
    from sklearn.linear_model import LinearRegression, Ridge, Lasso
    from sklearn.impute import SimpleImputer
    from sklearn.metrics import r2_score, mean_squared_error
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import OneHotEncoder, StandardScaler
```

```
# Train-test split (75-25), set seed to 10
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=10)

# Remove "object"-type features and SalesPrice from X
X_train_cont = X_train.select_dtypes(include=["float64", "int64"])
X_test_cont = X_test.select_dtypes(include=["float64", "int64"])

# Impute missing values with median using SimpleImputer
impute = SimpleImputer(strategy='median')

X_train_imputed = impute.fit_transform(X_train_cont)
X_test_imputed = impute.transform(X_test_cont)

# Scale the train and test data
ss = StandardScaler()

X_train_imputed_scaled = ss.fit_transform(X_train_imputed)
X_test_imputed_scaled = ss.transform(X_test_imputed)

# Create X_cat which contains only the categorical variables
X_train_cat = X_train.select_dtypes(include=["object"])
X_test_cat = X_test.select_dtypes(include=["object"])

# Fill nans with a value indicating that that it is missing
X_train_cat.fillna(value='missing', inplace=True)
X_test_cat.fillna(value='missing', inplace=True)


# OneHotEncode Categorical variables
ohe = OneHotEncoder(handle_unknown='ignore')

X_train_ohe = ohe.fit_transform(X_train_cat)
X_test_ohe = ohe.transform(X_test_cat)


# Combine categorical and continuous features into the final dataframe
X_train_all = pd.concat([pd.DataFrame(X_train_ohe.todense()),
                     pd.DataFrame(X_train_imputed_scaled)], axis = 1)

X_test_all = pd.concat([pd.DataFrame(X_test_ohe.todense()),
                    pd.DataFrame(X_test_imputed_scaled)], axis = 1)
```

```
    return X_train_all, X_test_all, y_train, y_test
```

1.12.1 Graph the training and test error to find optimal alpha values

Earlier we tested two values of alpha to see how it effected our MSE and the value of our coefficients. We could continue to guess values of alpha for our Ridge or Lasso regression one at a time to see which values minimize our loss, or we can test a range of values and pick the alpha which minimizes our MSE. Here is an example of how we would do this:

[194]:
```
X_train_all, X_test_all, y_train, y_test = preprocess(X, y)

train_mse = []
test_mse = []
alphas = []

for alpha in np.linspace(0, 200, num=50):
    lasso = Lasso(alpha=alpha)
    lasso.fit(X_train_all, y_train)

    train_preds = lasso.predict(X_train_all)
    train_mse.append(mean_squared_error(y_train, train_preds))

    test_preds = lasso.predict(X_test_all)
    test_mse.append(mean_squared_error(y_test, test_preds))

    alphas.append(alpha)
```

[195]:
```
print("X_train_all.shape: ", X_train_all.shape)
print("y_train.shape :", y_train.shape)
print("train_preds.shape:", train_preds.shape)
print()
print("X_test_all.shape: ", X_test_all.shape)
print("y_test.shape :", y_test.shape)
print("test_preds.shape:", test_preds.shape)
```

```
X_train_all.shape:  (1095, 296)
y_train.shape : (1095,)
train_preds.shape: (1095,)

X_test_all.shape:  (365, 296)
y_test.shape : (365,)
test_preds.shape: (365,)
```

[196]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

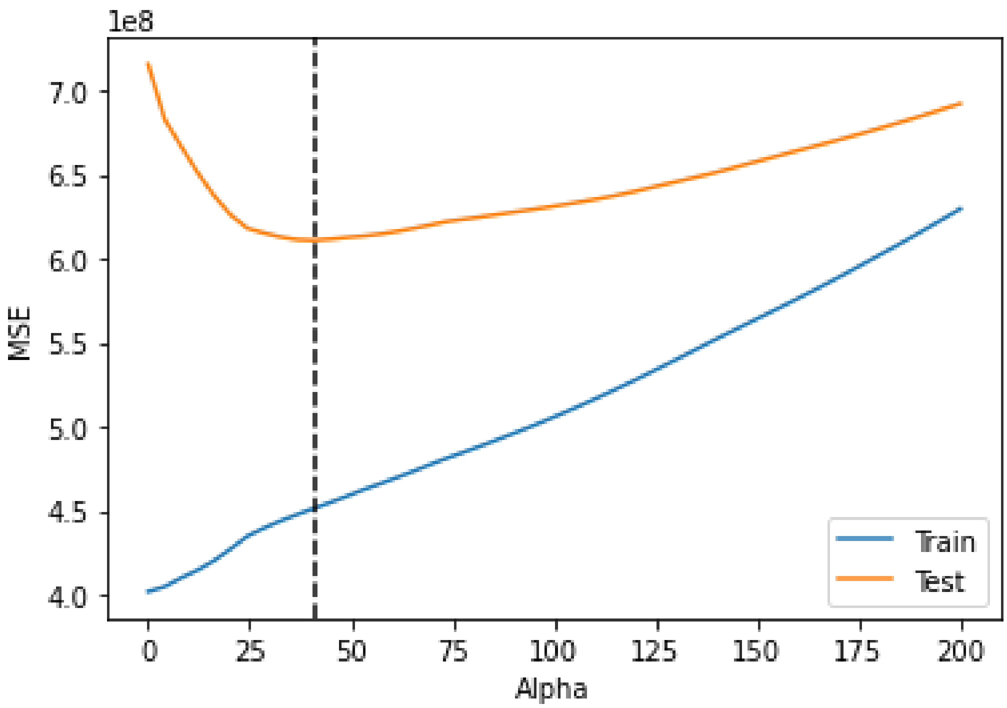
```
fig, ax = plt.subplots()
ax.plot(alphas, train_mse, label='Train')
ax.plot(alphas, test_mse, label='Test')
ax.set_xlabel('Alpha')
ax.set_ylabel('MSE')

# np.argmin() returns the index of the minimum value in a list
optimal_alpha = alphas[np.argmin(test_mse)]

# Add a vertical line where the test MSE is minimized
ax.axvline(optimal_alpha, color='black', linestyle='--')
ax.legend();

print(f'Optimal Alpha Value: {int(optimal_alpha)}')
```

```
Optimal Alpha Value: 40
```



Take a look at this graph of our training and test MSE against alpha. Try to explain to yourself why the shapes of the training and test curves are this way. Make sure to think about what alpha represents and how it relates to overfitting vs underfitting.

1.13 Summary

Well done! You now know how to build Lasso and Ridge regression models, use them for feature selection and find an optimal value for alpha.

```
[ ]:
```