CS675_homework1_lab_fall22_2

September 20, 2022

1 Homework 1: Lab

1.0.1 10 points total

1.0.2 Version 1.0

Mingtian Gao (mgao32), Wenxuan Lu (wlu15), Zhenhan Gan (zgan4)

Instructions: This notebook is intended to guide you through a regression task. Please answer all questions in this notebook (you will see TODO annotations for where to include your answers). At the beginning of each part, we will bullet the expected deliverables for you to complete. All questions can be answered in 1-4 sentences, unless otherwise noted.

1.1 Part 1: Regression

Things to do in this part: 1. Find a dataset according to the instructions below 2. Complete questions 1-2

Your task in this Lab is to explore a regression dataset from the UCI repository and train a linear regression model to solve the task.

1) Choose one datset from the UCI repository for regression task.
https://archive.ics.uci.edu/ml/datasets.php?format=&task=reg&att=&area=&numAtt=&numIns=&type=&sor

- Describe your dataset. State the features X and output variables Y you are interested in predicting clearly. What is the dimensionality of X and Y? Briefly, give some examples of what each dimension represents?

Answer: We chose the **Concrete Compressive Strength data set** with 1030 samples, which studied how concrete compressive strength, an important material in civial engineering, is related to age and ingredients.

X is a 1030 by 8 matrix with each row containing the cement component (unit: kg in a m^3 mixture), the blast furnace slag component (unit: kg in a m^3 mixture), the fly ash component (unit: kg in a m^3 mixture), the water component (unit: kg in a m^3 mixture), the superplasticizer component (unit: kg in a m^3 mixture), the coarse aggregate component (unit: kg in a m^3 mixture), the fine aggregate component (unit: kg in a m^3 mixture), and the age (unit: day). We are interested in predicting **Y**, which **is a 1030 by 1 vector** containing concrete compressive strength (unit: MPa), where each element corresponds to the concrete sample with features listed in the corresponding row in X. All the features and the responses are numeric.

For example, the first row in X is [540, 0, 0, 162, 2.5, 1040, 676, 28] and the first entry in Y is 79.98611076. This indicates that our frist example is the concrete with 540 kg/m^3 cement, 162 kg/m^3 water, 2.5 kg/m^3 superplasticizer, 1040 kg/m^3 coarse aggregate, and 676 kg/m^3 fine aggregate. Moreover, the cement was generated 28 days ago. This sample concrete has a compressive strength of 79.98611076 MPa.

- Print a few examples of datapoints to show what the raw data looks like.

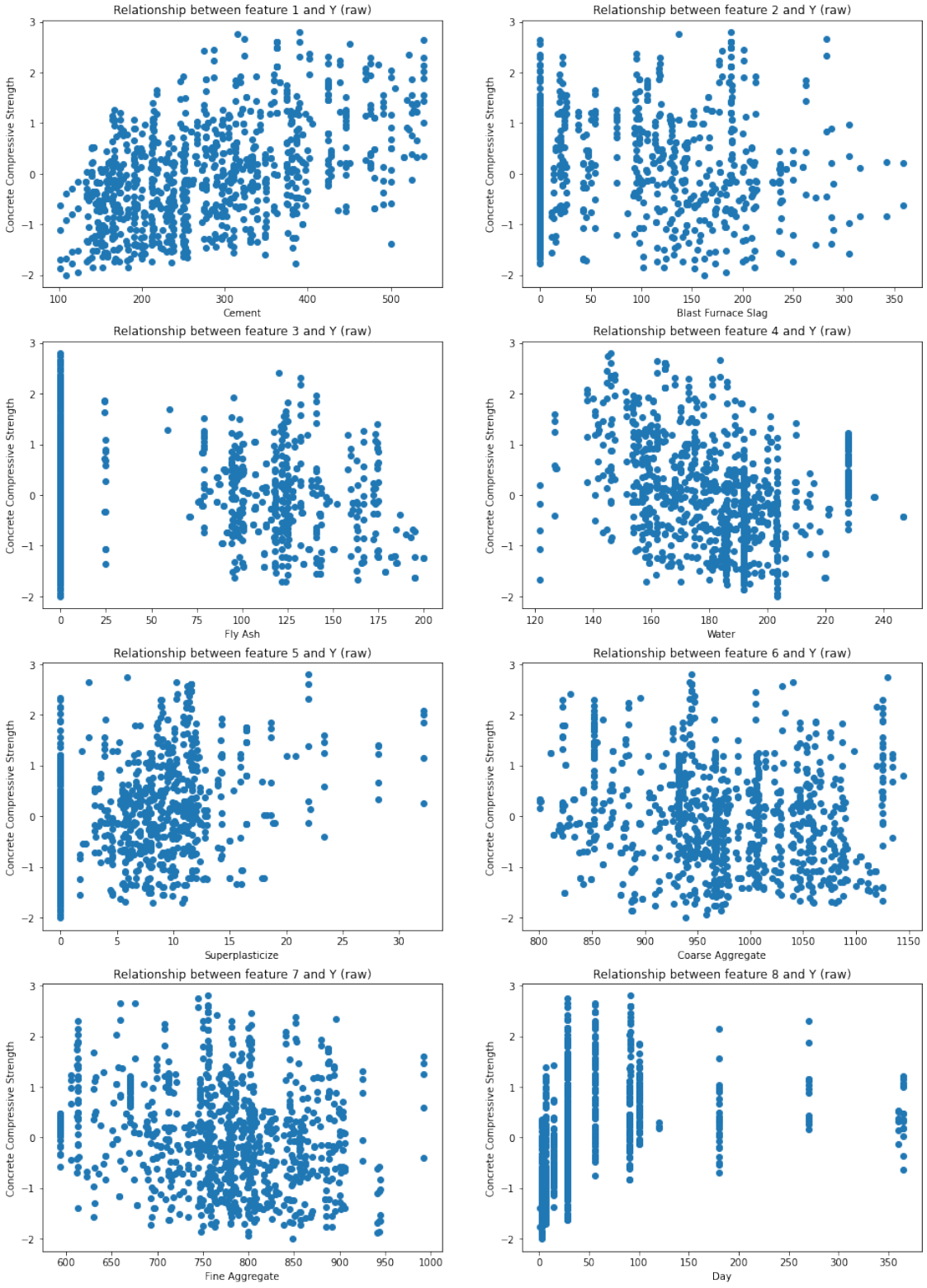
[1]:
```
##########################################################
import pandas as pd
data = pd.read_excel("Concrete_Data.xlsx",
                     header = None, names = ['cement','slag','ash', 'water',␣
 ↪'superplasticize','coarse','fine', 'Day','Y'])
##########################################################
print("Raw Data (first three rows of X and Y):")
X_raw = data.iloc[:, :8].values.copy()
y_raw = data['Y'].values.copy()
print (X_raw[:3]) #print the first three rows of your features
print (y_raw[:3]) #print the first three rows of your features
print()

##########################################################
# normalize the features
for col_names in data.columns:
   data[col_names] = (data[col_names] - data[col_names].mean()) /␣
 ↪data[col_names].std()

print("Normalized Data (first three rows of X and Y):")
X = data.iloc[:, :8].values
y = data['Y'].values
print (X[:3]) #print the first three rows of your features
print (y[:3]) #print the first three rows of your features
```

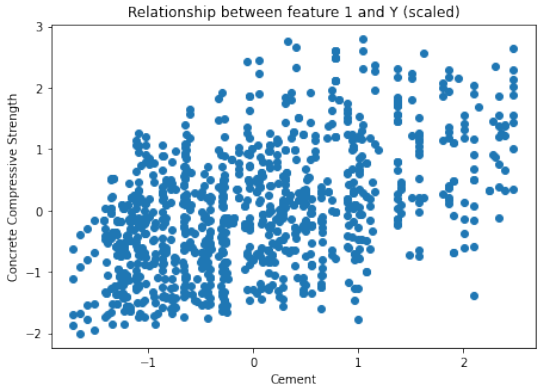
```
Raw Data (first three rows of X and Y):
[[ 540.     0.     0.   162.     2.5 1040.   676.    28. ]
 [ 540.     0.     0.   162.     2.5 1055.   676.    28. ]
 [ 332.5  142.5    0.   228.     0.   932.   594.   270. ]]
[79.98611076 61.88736576 40.26953526]

Normalized Data (first three rows of X and Y):
[[ 2.47671465 -0.85647025 -0.84672071 -0.9162182  -0.61992414  0.86274101
  -1.21706721 -0.27959729]
 [ 2.47671465 -0.85647025 -0.84672071 -0.9162182  -0.61992414  1.05565758
  -1.21706721 -0.27959729]
 [ 0.49120441  0.79514635 -0.84672071  2.17431083 -1.03843983 -0.5262583
  -2.23982446  3.55134048]]
[2.64390777 1.56051901 0.26647821]
```

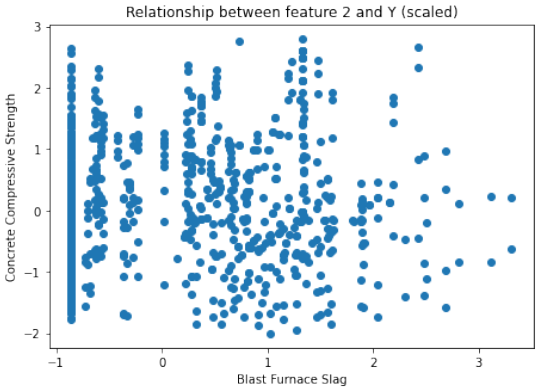
- Create 1 visualization that helps you understand the data. Below is an example from the Iris

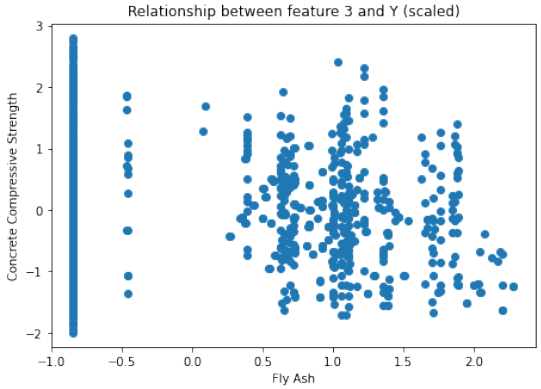
dataset. You will replace this for your regression dataset. One idea may be to create scatter plots of each feature in X with the output variable Y (the dependent variable).

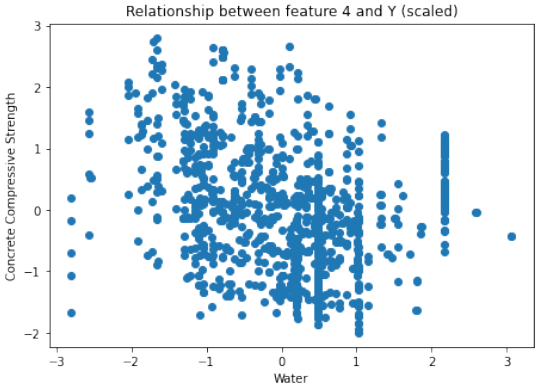
[2]:
```
import matplotlib.pyplot as plt
import seaborn as sns

feature_names = ['Cement','Blast Furnace Slag', 'Fly␣
 ↪Ash','Water','Superplasticize','Coarse Aggregate','Fine␣
 ↪Aggregate','Day','Concrete Compressive Strength (Y)']
fig = plt.figure(figsize =(16, 23))
for i in range(8):
    ax = fig.add_subplot(4, 2, i+1)
    ax.scatter(X_raw[:,i], y)
    ax.set_xlabel(feature_names[i])
    ax.set_ylabel('Concrete Compressive Strength')
    ax.set_title("Relationship between feature {} and Y (raw)".format(i+1))
plt.show()

fig = plt.figure(figsize =(16, 23))
for i in range(8):
    ax = fig.add_subplot(4, 2, i+1)
    ax.scatter(X[:,i], y)
    ax.set_xlabel(feature_names[i])
    ax.set_ylabel('Concrete Compressive Strength')
    ax.set_title("Relationship between feature {} and Y (scaled)".format(i+1))
plt.show()

corr = data.iloc[:, :].corr()
sns.heatmap(corr,
            xticklabels=feature_names,
            yticklabels=feature_names)
plt.show()
```

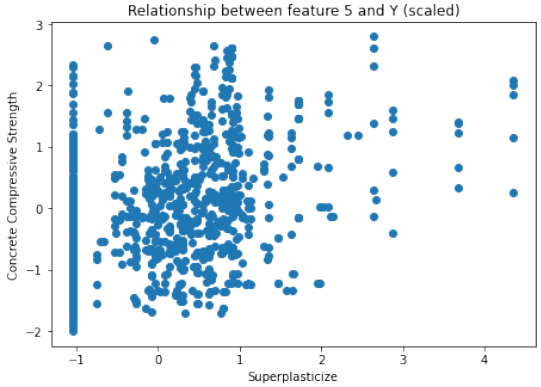

Relationship between feature 1 and Y (raw)
Concrete Compressive Strength
Cement
Relationship between feature 2 and Y (raw)
Concrete Compressive Strength
Blast Furnace Slag
Relationship between feature 3 and Y (raw)
Concrete Compressive Strength
Fly Ash
Relationship between feature 4 and Y (raw)
Concrete Compressive Strength
Water
Relationship between feature 5 and Y (raw)
Concrete Compressive Strength
Superplasticize
Relationship between feature 6 and Y (raw)
Concrete Compressive Strength
Coarse Aggregate
Relationship between feature 7 and Y (raw)
Concrete Compressive Strength
Fine Aggregate
Relationship between feature 8 and Y (raw)
Concrete Compressive Strength
Day

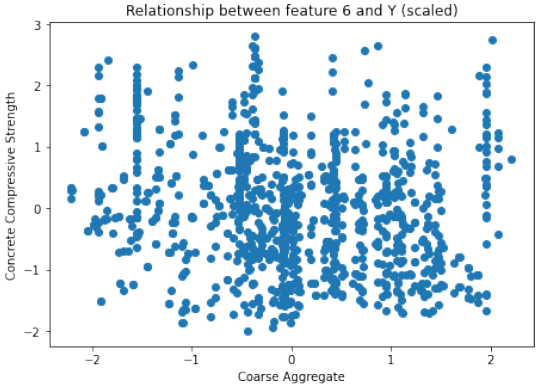

Relationship between feature 1 and Y (scaled)
Concrete Compressive Strength
Cement

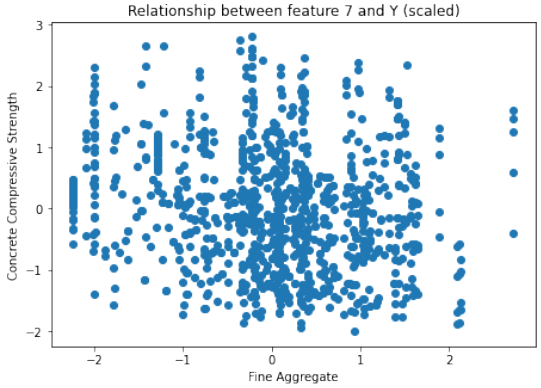

Relationship between feature 2 and Y (scaled)
Concrete Compressive Strength
Blast Furnace Slag

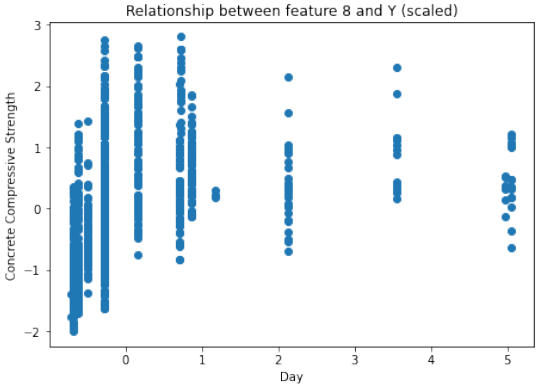

Relationship between feature 3 and Y (scaled)
Concrete Compressive Strength
Fly Ash

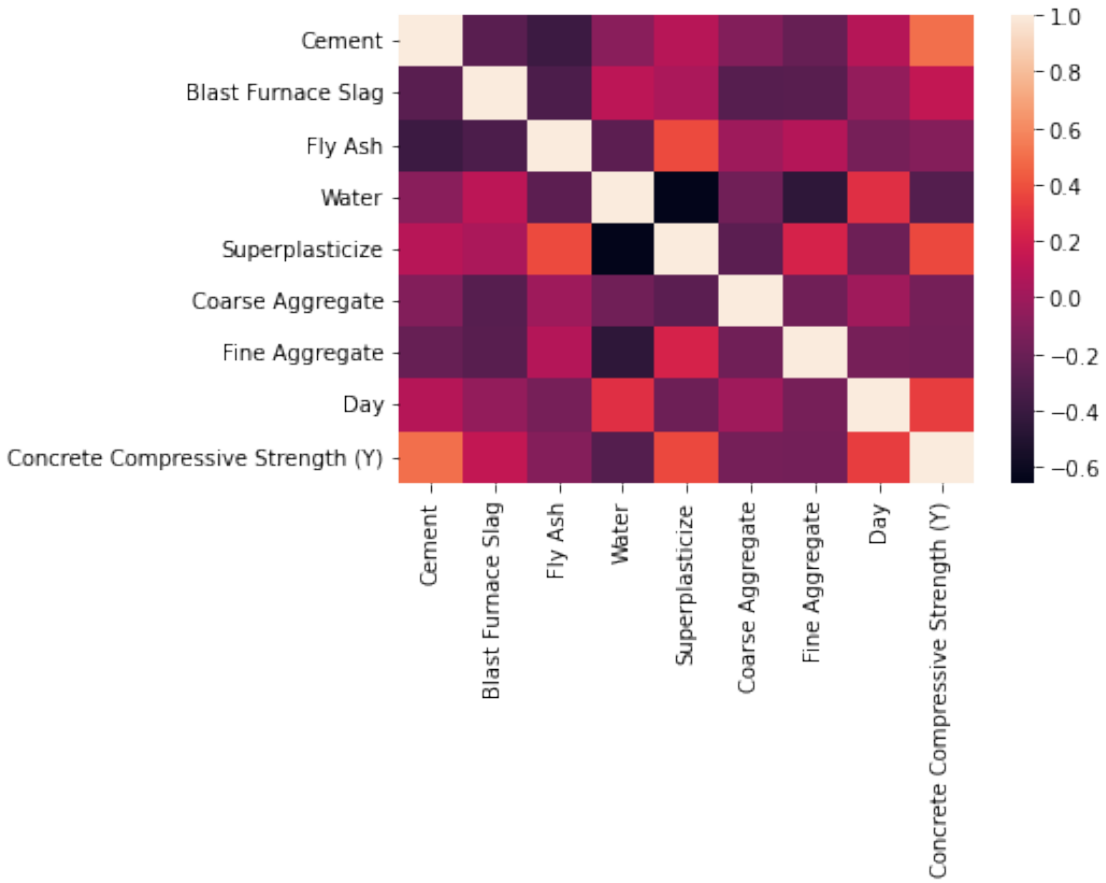

Relationship between feature 4 and Y (scaled)
Concrete Compressive Strength
Water


Relationship between feature 5 and Y (scaled)
Concrete Compressive Strength
Superplasticize


Relationship between feature 6 and Y (scaled)
Concrete Compressive Strength
Coarse Aggregate


Relationship between feature 7 and Y (scaled)
Concrete Compressive Strength
Fine Aggregate


Relationship between feature 8 and Y (scaled)
Concrete Compressive Strength
Day



- What insights do you gain from this visualization about your dataset? For example, in the above plot we see sepal length and sepal width are negatively correlated. In fact, sepalwidth is negatively correlated with all the features including the class label.

Answer: All the features are positive real values. The values of blast furnace slag, fly ash, and superplasticize for many samples are 0, which results in a concentration of points at 0 (in the raw data plots). Though the feature day can take values from 1 to 365, it only has 14 unique numbers in our dataset.

Correlation among features: except that the density of water component has a 0.66 negative correlation with the density of superplasticize component and a 0.45 negative correlation with the density of the fine aggregate component, the correlations among other features are low (below 0.4).

Correlation between output and each feature: We see that densities of day, superplasticizer, and cement are positively correlated with concrete strength while other features are negatively correlated with concrete strength.
Except that cement has a positive 0.45 correlation with the output, other features have a low correlations with it.

2) Train a linear regression model

- Create a train and test split for your dataset by diving the dataset into a 75-25 train-test split. If your chosen dataset already comes with a train-test split use that instead. Below is code for implementing a 75-25 split.

Note: We **used the normalized dataset** because normalization would make all features to be at a comparable scale. Thus, the regularization would apply to the coefficients of all features equally. Moreoever, **we used a 80-20 split** instead of a 75-25 split because our data set is small. We want to keep the test set the same as the one in later questions so that the results would be comparable.

[3]:
```
from sklearn.model_selection import train_test_split

# Train Test Split
X_train_all, X_test, y_train_all, y_test = train_test_split(X, y, test_size=0.
 ↪20, random_state=1)
print(f'Training examples: {X_train_all.shape[0]}\nTesting examples: { X_test.
 ↪shape[0]}')
```

```
Training examples: 824
Testing examples: 206
```

- Now that you have your train and test splits, it is time to train a linear regression model for making predictions on future data! You are encouraged to use popular packages such as sklearn; you are not expected to implement any of these algorithms yourself. We use sklearn's Linear Regression algorithm below.

[4]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

reg = LinearRegression().fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
```

```
Goodness of fit given by coefficient of determination is 0.5209616457703193
Mean Squared Error is 0.4474833022738864
```

- Explain what the MSE metric is? What value is expected for a good fit? Report your value on the above dataset.

Answer: MSE, the mean squared error, is the averaged squared distance between the true y (y_test) and the predicted y (y_hat). It measures how good a regression line fits to a set of points. The lower the MSE the better fit the model is. A MSE of 0 means that the model is a perfect fit.

Our model has an MSE of around 0.44748 after data normalization.

1.2 Part 2: Regularization

Things to do in this part: 1. Find a dataset according to the instructions below 2. Complete questions 3-4

3) Using the same dataset as above, try ridge (l2 regularizatin on the weights) and lasso (l1 regularization on the weights). Below is sample code for doing the same for ridge regression. You are free to use this. You will implement your own version of l1 regularization (lasso). You are free to use built-in implementation for scikit-learn or any other machine learning library.

[5]:
```
from sklearn.linear_model import Ridge

alpha = 1
print("Alpha = {}".format(alpha))
reg = Ridge(alpha=alpha).fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
print()

alpha = 8
print("Alpha = {}".format(alpha))
reg = Ridge(alpha=alpha).fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
print()

alpha = 500
print("Alpha = {}".format(alpha))
reg = Ridge(alpha=alpha).fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
```

```
Alpha = 1
Goodness of fit given by coefficient of determination is 0.5214863803099925
Mean Squared Error is 0.44699313286979414

Alpha = 8
Goodness of fit given by coefficient of determination is 0.5240019555486763
Mean Squared Error is 0.4446432627498225

Alpha = 500
Goodness of fit given by coefficient of determination is 0.500955306288799
Mean Squared Error is 0.46617179094824157
```

[6]:
```
from sklearn.linear_model import Lasso

alpha = 0.001
print("Alpha = {}".format(alpha))
reg = Lasso(alpha=alpha).fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
print()

alpha = 0.005
print("Alpha = {}".format(alpha))
reg = Lasso(alpha=alpha).fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
print()

alpha = 1
print("Alpha = {}".format(alpha))
reg = Lasso(alpha=alpha).fit(X_train_all, y_train_all)
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f"The coefficients are {reg.coef_}, which implies r square is 0,")
print("here r square value is negative because the iterative method used is␣
 ↪just an approximation.")
print(f'Mean Squared Error is {mse}')
print()
```

```
Alpha = 0.001
Goodness of fit given by coefficient of determination is 0.5216996915179427
Mean Squared Error is 0.44679387282536803

Alpha = 0.005
Goodness of fit given by coefficient of determination is 0.5238235191072276
Mean Squared Error is 0.44480994528653495

Alpha = 1
Goodness of fit given by coefficient of determination is -0.016127174062290583
The coefficients are [ 0.  0. -0. -0.  0. -0. -0.  0.], which implies r square
is 0,
```

```
here r square value is negative because the iterative method used is just an
approximation.
Mean Squared Error is 0.949193189574158
```

- For both l1 and l2 regularization change the regularization hyperparameter (for example alpha hyperparameter in the above cell code for ridge regression) and report changes. What do you observe? Can you explain your observation in terms of model complexity, overfitting and underfitting?

As alpha increases from 0 by small step, MSE first decreases and then increases for both l1 and l2 regression. When there is no regularization, i.e., alpha = 0, the model is overfitting our training data (actually MSE of l1 drops much less than that of l2 and starts increasing in an early point, indicating that l1 has little overfitting problem initially). As we increase alpha, model complexity decreases and the decrease in variance is greater than increase in bias square, which makes MSE decrease. But when alpha keeps increasing, increase in bias square is greater than decrease in variance, thus MSE increases and the model becomes underfitting the data.

4) Model Selection

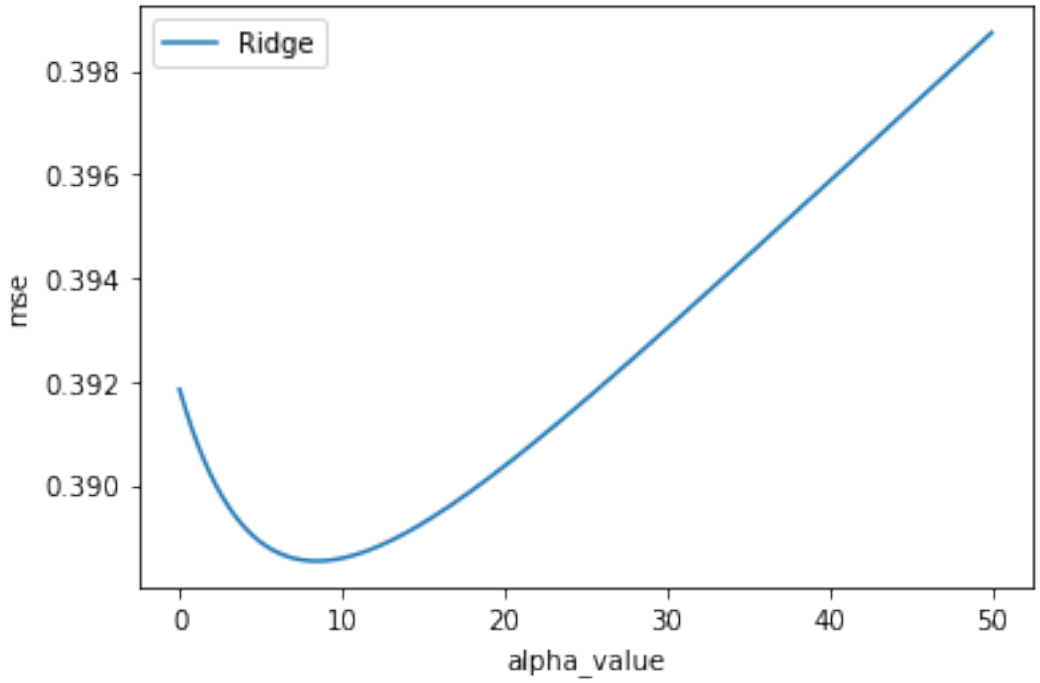
- In this part we would do model selection using different hyperparameters for regularizaiton we experimented with in the question 3. We would first split the dataset into train-val-test sets. Feel free to experiment with the below code to do so.

[7]:
```
from sklearn.model_selection import train_test_split

#Divide dataset into a 80-20 random split into training and testing set
X_train_all, X_test, y_train_all, y_test = train_test_split(X, y, test_size=0.
↪2, random_state=1)

#Divide the train set further into a 75-25 random split into train-val. If your␣
↪dataset is small you might want to reduce the test-size parameter for val␣
↪set.
X_train, X_val, y_train, y_val = train_test_split(X_train_all, y_train_all,␣
↪test_size=0.25, random_state=1) # 0.25 x 0.8 = 0.2
```

- Do a grid-search for best hyperparameters by varying the hyperparameter controlling the l1 and l2 regularization of the weights in liner regression. Include the case where alpha=0 corresponding to vanilla linear regression in your experiment. For each hyperparameter, train your model and evaluate the performance on the validation set. Report this in a plot with x-axis being different hyperparameter values for the regularization coefficient and the y-axis being the validation MSE metric obtained. You will report 2 curves one for l1 regularization and one for l2.

[8]:
```
import numpy as np
from sklearn.linear_model import LinearRegression

result_Ridge = []
alpha = np.arange(0, 50, 0.01)
```

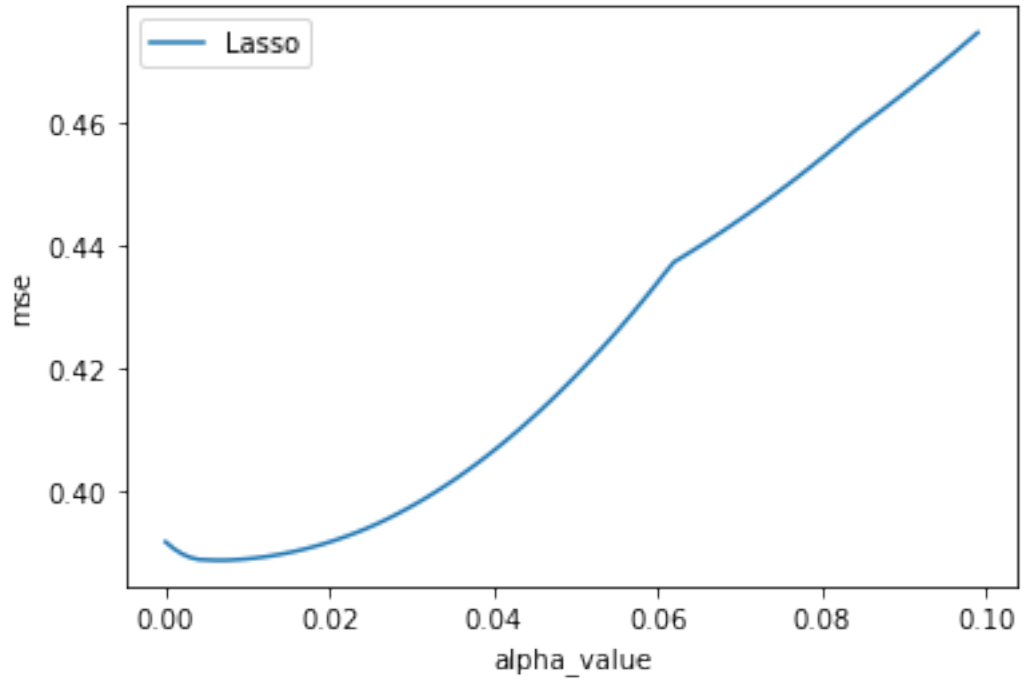
```
for i in alpha:
    reg = Ridge(alpha=i).fit(X_train, y_train)
    y_hat = reg.predict(X_val)
    mse = mean_squared_error(y_val, y_hat)
    result_Ridge.append(mse)
    #print(reg.coef_)

plt.plot(alpha, result_Ridge, label = 'Ridge')
plt.xlabel("alpha_value")
plt.ylabel("mse")
alpha_Ridge = alpha[np.argmin(result_Ridge)]

plt.legend()
plt.show()
print("Best alpha for L2 = ", alpha_Ridge)
```




```
Best alpha for L2 =  8.44
```

[9]:
```
result_Lasso = []
alpha = np.arange(0, 0.1, 0.001)
for i in alpha:
    if i == 0:
        reg = LinearRegression().fit(X_train, y_train)
    else:
        reg = Lasso(alpha=i).fit(X_train, y_train)
```

```
    y_hat = reg.predict(X_val)
    mse = mean_squared_error(y_val, y_hat)
    result_Lasso.append(mse)
    #print(reg.coef_)

plt.plot(alpha, result_Lasso, label = 'Lasso')
plt.xlabel("alpha_value")
plt.ylabel("mse")
plt.legend()
plt.show()

alpha_Lasso = alpha[np.argmin(result_Lasso)]
print("Best alpha for L1 = ", alpha_Lasso)
```




```
Best alpha for L1 =  0.007
```

- **bold text** Take your best performing hyperparameters from the previous question as your selected model and report the performance on test set. Is it better than what you reported in question 2?

[10]:
```
reg = Ridge(alpha=alpha_Ridge).fit(np.concatenate((X_train, X_val)), np.
 ↪concatenate((y_train, y_val)))
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
```

```
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
```

```
Goodness of fit given by coefficient of determination is 0.5241177163355438
Mean Squared Error is 0.4445351273182357
```

[11]:
```
if alpha_Lasso == 0:
    reg = LinearRegression().fit(X_train, y_train)
else:
    reg = Lasso(alpha=alpha_Lasso).fit(np.concatenate((X_train, X_val)), np.
 ↪concatenate((y_train, y_val)))
y_hat = reg.predict(X_test)
r_squared = reg.score(X_test, y_test)
print(f'Goodness of fit given by coefficient of determination is {r_squared}')
mse = mean_squared_error(y_test, y_hat)
print(f'Mean Squared Error is {mse}')
```

```
Goodness of fit given by coefficient of determination is 0.5257408871058924
Mean Squared Error is 0.44301887750221003
```

Answer: Mean Squared Error in question 2 is about 0.44748, which is larger than the Mean Squared Errors of Ridge and Lasso after regularization, so we conclude that they are better than the regression model in question 2.

This concludes homework lab 1!