Sushanth Ravichandran (sr56925)

Sankeerth Viswanadhuni (vps386)

HW 1

Due: 11:59 PM CT, September 13

Total points: 75

Your homework should be written in a Python notebook and then exported to a PDF. You may work in groups of two if you wish. Only one student per team needs to submit the assignment on Gradescope. But be sure to include name and UT EID for both students.

Also, please make sure your code runs and the graphics (and anything else) are displayed in your notebook and PDF before submitting. (%matplotlib inline)

Question 1: AWS SageMaker (10 pts):

Read the article titled "10 reasons why Amazon SageMaker is great for machine learning" on Amazon SageMaker and answer the following questions:

1. In two paragraphs and in your own words, highlight some key benefits that a platform like Amazon SageMaker offers for the development of a machine learning (ML) application. (5 pts)
2. Use a large language model (LLM) of your choice to generate an alternative response to the above question. Be sure to include the name of the LLM used and the specific (sequence of) prompt(s) you provided. (3 pts)
3. Compare your explanation with the output from the LLM, and summarize your observations regarding the differences between the two answers. (2 pts)

Answer

1.)Amazon SageMaker is a user-friendly platform that makes building, training, and deploying machine learning models easy. It can be easily integrated with other AWS services, making it easy to build complete machine learning workflows. AutoML capabilities of sagemakers helps users with identifying and building the suitable model based on the dataset provided. This feature helps a lot of small companies to build their ML capabilities very easily.

One Click training and deployment features enables to build and deploy models with a single click helping users to focus more on building and refining theie models. Sagemaker also helps with resource scaling and security aspects of Model building and deployment by implementing auto sacling techniques based on workload

2.) For this task, I used OpenAI’s GPT-4 to generate an alternative response. Here's the sequence of prompts I used to get the response:

Prompt 1: “Summarize the key benefits and features of Amazon SageMaker in two paragraphs.

GPT-4 Generated Response: Amazon SageMaker simplifies the process of building, training, and deploying machine learning models by offering a fully managed environment. Its one-click training and deployment feature allows data scientists and developers to quickly set up models without extensive manual configuration. SageMaker also provides a range of built-in algorithms, frameworks like TensorFlow and PyTorch, and tools for feature engineering and bias detection (via SageMaker Clarify). This ease of use helps reduce the complexities typically involved in machine learning workflows, allowing teams to focus on improving model performance rather than managing infrastructure.

Furthermore, SageMaker's integration with other AWS services, such as S3 for data storage, EC2 for scalable compute power, and CloudWatch for monitoring, makes it a powerful tool for handling large-scale machine learning tasks. It also offers AutoML capabilities for automatic model generation and fine-tuning, along with built-in security features like encryption and role-based access control. SageMaker’s advanced debugging and monitoring tools, such as SageMaker Debugger, help teams identify and resolve issues during training, further enhancing the platform’s flexibility and scalability.

3.) I highlighted specific features of Amazon SageMaker, like its managed services, one-click deployment, and tools for auto Model building. In my explaination i also covered about for whom it could be useful based on specific capablities. The response from the LLM took a higher-level approach, focusing on ease of use and scalability, while briefly touching on security and debugging. It grouped features into broader themes for a more streamlined summary.

Question 2: Google Flu Trends (10 pts)

The article "The Parable of Google Flu: Traps in Big Data Analysis" (kept in Canvas --> Modules --> Resources) describes a high-profile (and embarrassingly failed) project done by Google, highlighting the phenomena of data drift and the importance of transparency, among other key issues that an ML project can face.

Read this article and answer the following questions

(i) Briefly describe two important causes of "data drift" in the flu prediction problem that are mentioned in the article (5 pts)

(ii) The last section highlights the importance of transparency. Express in your words, why is transparency important for building data science and AI projects? (You can check this article as a helpful reference: https://hbr.org/2022/06/building-transparency-into-ai-projects or you can look for other sources and cite them in your answer) (5 pts)

Answer

(i) Data drift in flu prediction can stem from changes in user behavior and shifts in healthcare practices. For example, as people become more familiar with flu symptoms or online tracking tools, their search patterns may evolve, affecting the data the model uses. Similarly, updates in healthcare practices, such as new vaccines or treatments, can alter how flu cases are reported and managed. These changes can impact the model's accuracy by altering the data it relies on.

(ii) The article underscores the crucial role of transparency in AI projects, emphasizing that it's an ongoing process rather than a final step. Transparency helps build trust with stakeholders by reducing errors, sharing responsibility, and allowing for proper oversight. It ensures that all parties—from data scientists to executives—understand how results are achieved, which helps in ethical decision-making and minimizes risks. Clear communication throughout the project fosters better understanding and trust in the AI's outcomes.

Question 3: Maximum Likelihood Estimate (10 pts)

Consider the following probability density function (pdf) of a random variable X that takes value $x \in [0, \infty)$

$$f(x; b) = \frac{x}{Kb^2}\exp(-\frac{x}{b})$$

where b is the parameter of the density function, and K is a suitable normalizing constant. The (unscaled) pdfs with $b = 0.1, 0.5, 1$ are plotted below to help you visualize how this family of pdfs looks like.

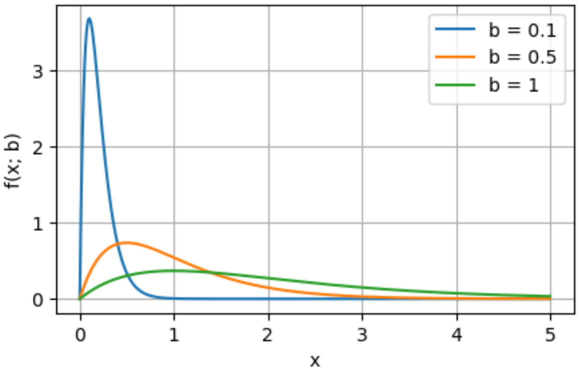
Suppose we observe 6 values obtained by sampling i.i.d. from the pdf described above:

$$\{x_1, x_2, x_3, x_4, x_5, x_6\} = \{0.1, 0.5, 1, 0.4, 2, 1.4\}$$

Answer the following questions:

1. What is the log-likelihood of observing a set of 6 samples drawn i.i.d from the pdf mentioned above (give an algebraic expression that is valid for any set of 6 samples)? (5 points)
2. What is the maximum likelihood estimate for the parameter b? (5 points)

```
import numpy as np
import matplotlib.pyplot as plt

def pdf(x, b):
    return (x / (b**2)) * np.exp(-x / b) # K = 1

b_values = [0.1, 0.5, 1]

x = np.linspace(0, 5, 500)

plt.figure(figsize=(5, 3))
for b in b_values:
    plt.plot(x, pdf(x, b), label=f'b = {b}')

plt.xlabel('x')
plt.ylabel('f(x; b)')
plt.legend()
plt.grid(True)
plt.show()
```



Answer

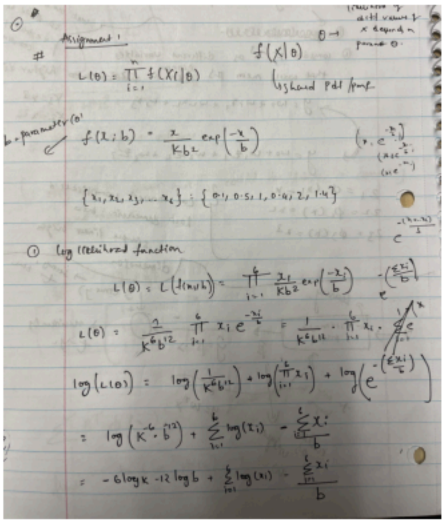
1.) Log L(b)=log(xi)-6*Log K-12* Log(b)-(1/b)*xi where xi is the sum of all values from x=x1 to x6

2.) Take the derivative of above algebraic equation and equate it to 0; subistitute values of xi and b=0.45

```
from google.colab import files
uploaded = files.upload()
```

```
Choose files 2 files
• Q3_1.jpg(image/jpeg) - 1219162 bytes, last modified: 13/09/2024 - 100% done
• Q3_2.jpg(image/jpeg) - 1153066 bytes, last modified: 13/09/2024 - 100% done
Saving Q3_1.jpg to Q3_1.jpg
Saving Q3_2.jpg to Q3_2.jpg
```

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Load the images
img1 = mpimg.imread('Q3_1.jpg')
img2 = mpimg.imread('Q3_2.jpg')

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Display the first image in the first subplot
axes[0].imshow(img1)
axes[0].axis('off')  # Optional: Turn off axis for the first image

# Display the second image in the second subplot
axes[1].imshow(img2)
axes[1].axis('off')  # Optional: Turn off axis for the second image

# Display the figure
plt.show()
```



Assignment 1

$L(\theta) = \prod_{i=1}^{n} f(X_i|\theta)$ $f(x|\theta)$ ↳ Joint pdf/pmf

$\theta \rightarrow$ diff values of x depend on param θ.

b - parameter 'θ'

$$f(x;b) = \frac{x}{Kb^2}\exp\left(\frac{-x}{b}\right)$$

$\{x_1, x_2, x_3, \ldots x_6\} : \{0.1, 0.5, 1, 0.4, 2, 1.4\}$

① log likelihood function

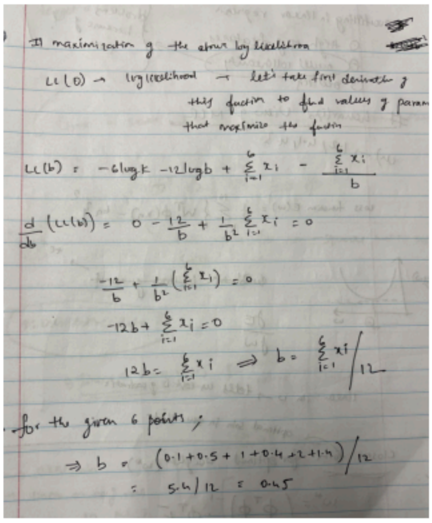
$$L(\theta) = L(f(x_i|b)) = \prod_{i=1}^{6} \frac{x_i}{Kb^2}\exp\left(\frac{-x_i}{b}\right)$$

$$L(\theta) = \frac{1}{K^6 b^{12}} \prod_{i=1}^{6} x_i e^{-\frac{x_i}{b}} = \frac{1}{K^6 b^{12}} \cdot \prod_{i=1}^{6} x_i \cdot e^{-\left(\frac{\sum x_i}{b}\right)}$$

$$\log(L(\theta)) = \log\left(\frac{1}{K^6 b^{12}}\right) + \log\left(\prod_{i=1}^{6} x_i\right) + \log\left(e^{-\left(\frac{\sum x_i}{b}\right)}\right)$$

$$= \log\left(K^{-6} \cdot b^{-12}\right) + \sum_{i=1}^{6}\log(x_i) - \frac{\sum_{i=1}^{6} x_i}{b}$$

$$= -6\log K - 12\log b + \sum_{i=1}^{6}\log(x_i) - \frac{\sum_{i=1}^{6} x_i}{b}$$



II maximization of the above log likelihood

$LL(\theta) \rightarrow$ log likelihood $\rightarrow$ let's take first derivative of this function to find values of param that maximize the function

$$LL(b) = -6\log K - 12\log b + \sum_{i=1}^{6} x_i - \frac{\sum_{i=1}^{6} x_i}{b}$$

$$\frac{d}{db}(LL(b)) = 0 - \frac{12}{b} + \frac{1}{b^2}\sum_{i=1}^{6} x_i = 0$$

$$\frac{-12}{b} + \frac{1}{b^2}\left(\sum_{i=1}^{6} x_i\right) = 0$$

$$-12b + \sum_{i=1}^{6} x_i = 0$$

$$12b = \sum_{i=1}^{6} x_i \Rightarrow b = \sum_{i=1}^{6} x_i \Big/ 12$$

for the given 6 points;

$$\Rightarrow b = (0.1 + 0.5 + 1 + 0.4 + 2 + 1.4)/12$$

$$= 5.4/12 = 0.45$$

Question 4: Linear Regression (10 pts)

1. What is the difference between R-square and adjusted R square and why is it desirable to use the adjusted value? (4 pts)
2. Overfitting usually happens in complex models. Linear Regression is a fairly simple model. Could overfitting happen in Linear Regression? If so, please explain the scenario in which it could happen and how we can tackle it. (6 pts)

Answer

1.) R Square is the measure of variance in the dependent variable that can be predicted by the independent variables. It ranges from 0 to 1 and the R Squared value increases with increase in number of independent variables added even if the independent variables add no value. Therefore Adjusted R Square is calculated which takes into account the number of independent variables and can decrease in value if the addition of new parameters is of no value. Adjusted R Square can help reduce overfitting.

2.0verfitting can happen in linear regression as well. Some of the scenarios are as follows: a.) If there are unnecessarily high number of predictors compared to the number of data point. Then in this case a lot of noice will be captured. b.) If there are correlated predictors then small changes to the predictors could yield very different results c.) R Square value being high but adjusted R Square value being low due to unnecessar predictors which fetch noice that do not represent the data well overall

Question 5: Ridge/ Lasso Regression (35 pts)

This is a programming question. Please read through each subpart of this question carefully. You are required to add lines of code as specified in the code cells. Please carefully read through the comments in the code cells to identify what code is to be written, where it is to be written and how many lines of code are required. Code is to be added between the **## START CODE** ## and **## END CODE ##** comments and in place of the keyword **None**. In certain cases, the number of lines of code that are to be written will be specified. For example, **## START CODE ## (1 line of code)** specifies that only 1 line of code is to be added between the ## START CODE ## and ## END CODE ## comments. In case there is no information on the required number of lines, you are allowed to add any number of lines of code.

The following question covers a dataset for house cost and linear models in python. The categorical variables and rows with missing variables are removed to make it easier to run the models.

NOTE

- Only use the following code block if you are using Google Colab. If you are using Jupyter Notebook, please ignore this code block. You can directly upload the file to your Jupyter Notebook file systems.
- It will prompt you to select a local file. Click on “Choose Files” then select and upload the file. Wait for the file to be 100% uploaded. You should see the name of the file once Colab has uploaded it.

```
from google.colab import files
uploaded = files.upload()
```

Imports required

```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn.metrics import r2_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
pd.options.mode.chained_assignment = None

df = pd.read_csv('house_cost_data.csv')
X = df.drop(['house_cost'],axis=1)
Y = df['house_cost']
```

```
# Show you all the columns in this file
df.columns
```

```
Index(['house_cost', 'num_of_beds', 'num_of_baths', 'living_area', 'sqft_lot',
       'floors', 'condition', 'grade', 'sqft_above', 'sqft_basement',
       'built_year', 'renovation_year', 'latitude', 'longitude',
       'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

```
# Show you the first 5 rows in this file
df.head()
```

	house_cost	num_of_beds	num_of_baths	living_area	sqft_lot	floors	condition	grade	sqft_above	sqft_basement	built
0	221900	3	1.00	1180	5650	1.0	3	7	1180	0	
1	538000	3	2.25	2570	7242	2.0	3	7	2170	400	
2	180000	2	1.00	770	10000	1.0	3	6	770	0	
3	604000	4	3.00	1960	5000	1.0	5	7	1050	910	
4	510000	3	2.00	1680	8080	1.0	3	8	1680	0	

Part-1: (2 pts)

Split the data into a training set (75% of data) and a test set (25% of data), using the train_test_split function with random_state = 50.

```
##  START CODE  ## (1 line of code)
X_train, X_test, y_train, y_test =  train_test_split(X, Y, test_size=0.25, random_state=50)
##  END CODE    ##
```

Scale the data (not including target) so that each of the independent variables would have zero mean and unit variance. You can use the sklearn.preprocessing.scale function for this.

```
from sklearn.preprocessing import scale
##  START CODE  ## (2 lines of code)
X_train_scaled = scale(X_train)
X_test_scaled = scale(X_test)
##  END CODE    ##

y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
```

Print the first 5 rows of the training set after scaling

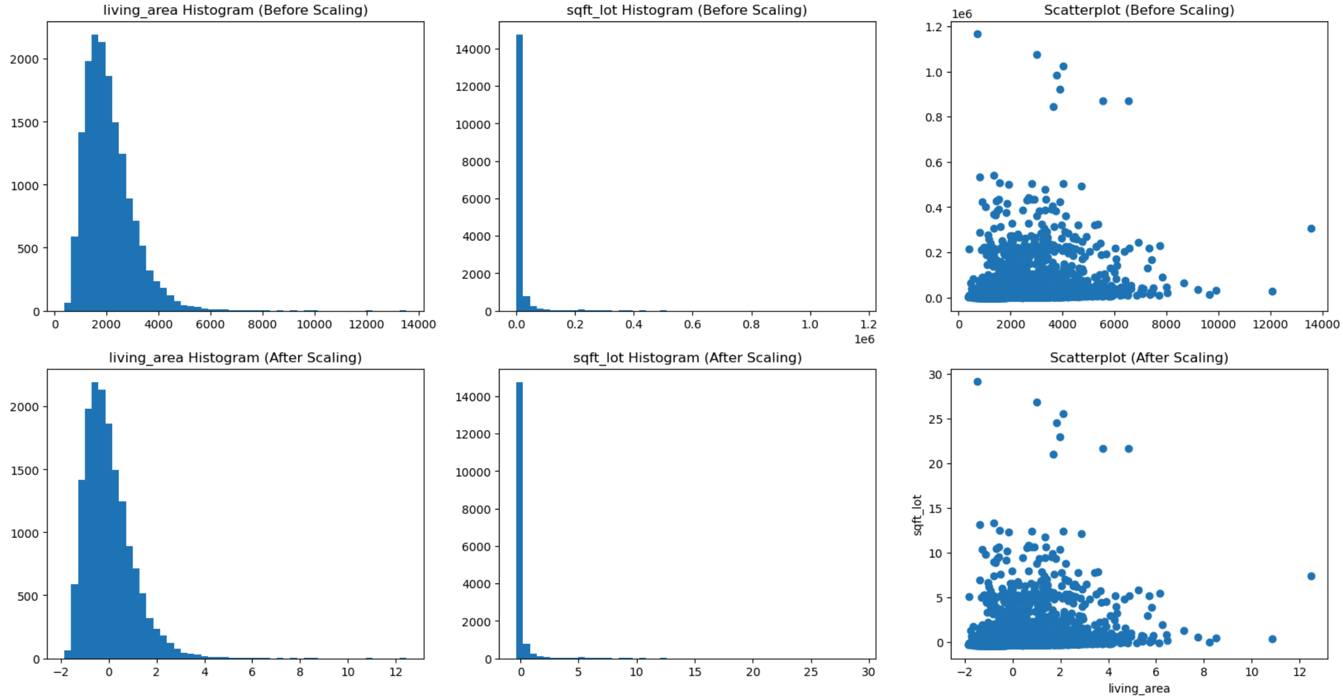
```
##  START CODE  ##
X_train_scaled[:5,]
##  END CODE    ##
```

```
array([[-1.46128799, -1.45010618, -1.36293279, -0.22520747, -0.92013997,
         0.90694939, -1.41249264, -1.16171819, -0.65460814, -0.58245852,
        -0.20780192,  1.34578638, -0.85717047, -0.94773519, -0.24597874],
       [-1.46128799, -1.45010618, -1.37381314,  0.02822559, -0.92013997,
         0.90694939, -2.26332668, -1.1737881 , -0.65460814, -1.63697323,
        -0.20780192, -2.57128162,  1.48741095, -0.1585854 ,  0.05298474],
       [-0.40089416, -1.45010618, -1.08004373, -0.11690425, -0.92013997,
         0.90694939, -1.41249264, -0.8479005 , -0.65460814, -0.71852494,
        -0.20780192,  1.54571781, -0.33300423, -0.11474375, -0.08704936],
       [-1.46128799,  0.49115341, -0.80803501, -0.35221507,  0.93675106,
        -0.63110697,  0.28917544, -1.02894917,  0.24778822,  1.25443807,
        -0.20780192,  0.0386535 , -1.2255035 , -0.75775469, -0.41790328],
       [-0.40089416, -0.47947639, -0.20961583, -0.28068787, -0.92013997,
        -0.63110697, -0.5616586 , -0.60650228,  0.6989864 ,  0.09787355,
        -0.20780192,  0.88745983, -1.19008687, -0.97696296, -0.32869222]])
```

Select any two variables. See how their histograms and scatterplots compare before and after scaling.

```
##  START CODE  ##
var1 = 'living_area'
var2 = 'sqft_lot'

plt.figure(figsize=(20, 10))

# Histograms before scaling
plt.subplot(2, 3, 1)
plt.hist(X_train[var1], bins=50)
plt.title(f'{var1} Histogram (Before Scaling)')

plt.subplot(2, 3, 2)
plt.hist(X_train[var2], bins=50)
plt.title(f'{var2} Histogram (Before Scaling)')

# Scatteplot before scaling
plt.subplot(2, 3, 3)
plt.scatter(X_train[var1], X_train[var2])
plt.title('Scatterplot (Before Scaling)')

# Histograms after scaling
plt.subplot(2, 3, 4)
plt.hist(X_train_scaled[:, X_train.columns.get_loc(var1)], bins=50)
plt.title(f'{var1} Histogram (After Scaling)')

plt.subplot(2, 3, 5)
plt.hist(X_train_scaled[:, X_train.columns.get_loc(var2)], bins=50)
plt.title(f'{var2} Histogram (After Scaling)')

# Scatterplot after scaling
plt.subplot(2, 3, 6)
plt.scatter(X_train_scaled[:, X_train.columns.get_loc(var1)],
            X_train_scaled[:, X_train.columns.get_loc(var2)])
plt.title('Scatterplot (After Scaling)')
plt.xlabel(var1)
plt.ylabel(var2)


## END CODE ##
```

```
Text(0, 0.5, 'sqft_lot')
```



```
X_train.columns.get_loc(var1)
```

```
2
```

Part-2: (5 pts)

Use `sklearn.linear_model.Lasso` and `sklearn.linear_model.Ridge` classes to do a 5-fold cross validation using sklearn's `KFold`. For the sweep of the regularization parameter, we will look at a grid of values ranging from $\alpha = 10^{-6}$ to $\alpha = 10^{6}$.In Python, you can consider this range of values as follows: `alpha = 10**numpy.linspace(-6, 6, 100)` so that you can generate 100 uniform values between -6 to 6 as power series.

Fit the 2 regression models (Lasso and Ridge) with scaled data and report the best chosen α based on cross validation as well as the corresponding scoring metric. The cross validation should happen on your training data using MSE as the scoring metric.

```
# Define number of folds
##  START CODE  ## (1 line of code)
n_folds = 5
##  END CODE  ##
```

```
# Create KFold from sklearn
##  START CODE  ## (1 line of code)
k_fold = KFold(n_splits=n_folds, shuffle=True, random_state=50)
##  END CODE    ##
```

```
#Define the alphas as defined in the question
##  START CODE  ## (1 line of code)
alphas = 10**np.linspace(-6, 6, 100)
##  END CODE    ##

lasso_avg_mse = {}
ridge_avg_mse = {}


from sklearn.linear_model import Lasso
import warnings
from sklearn.exceptions import ConvergenceWarning

# Suppress specific sklearn convergence warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)
#For each value of alpha and each fold compute the mean square error
#---------NOTE: This might take a while to run, so please be patient---------------#
#---------NOTE: There will be some convergence warning for smaller alphas, but you can ignore it---------------#

for alpha in alphas:

  #Instantiate a lasso model with the current alpha
  ##  START CODE  ## (1 line of code)
    lasso = Lasso(alpha=alpha)
  ##  END CODE    ##

    avg_mse = 0
    for k, (train, test) in enumerate(k_fold.split(X_train_scaled, y_train)):

    #Fit the scaled training data to the lasso model
    ## START CODE ## (1 line of code)
        lasso.fit(X_train_scaled[train], y_train[train])
    ## END CODE ##
        # Predict on the test set
        y_pred = lasso.predict(X_train_scaled[test])

    #Calculate the average mean sqaured error
    ##  START CODE  ## (1 line of code)
        avg_mse  += mean_squared_error(y_train[test], y_pred)
    ##  END CODE    ##

    # Take the average mean squared error as metric
        lasso_avg_mse[alpha] = avg_mse / n_folds


# Find the best value for alpha with minimum mean squared error
##  START CODE  ## (1 line of code)
best_alpha_lasso = min(lasso_avg_mse, key=lasso_avg_mse.get)
##  END CODE    ##

print("Best lasso alpha: {}".format(best_alpha_lasso))
```

```
Best lasso alpha: 533.6699231206302
```

```
from sklearn.linear_model import Ridge

#For each value of alpha and each fold compute the mean square error
for alpha in alphas:


  #Instantiate a ridge model with the current alpha
  ##  START CODE  ## (1 line of code)
    ridge_model = Ridge(alpha=alpha)
  ##  END CODE    ##
```

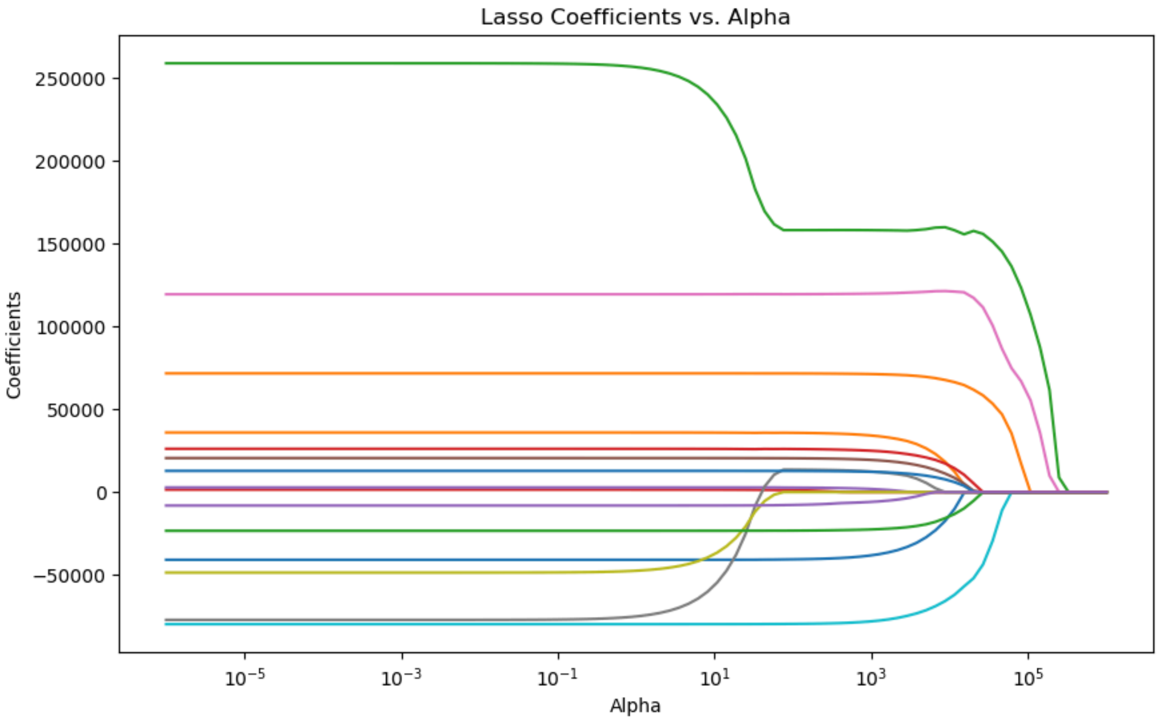
```
    avg_mse = 0

    for k, (train, test) in enumerate(k_fold.split(X_train_scaled, y_train)):

    #Fit the scaled training data to the ridge model
    ## START CODE ## (1 line of code)
        ridge_model.fit(X_train_scaled[train], y_train[train])
        # Predict on the test data
        y_pred = ridge_model.predict(X_train_scaled[test])
    ## END CODE ##

    #Calculate the average mean sqaured error
    ##  START CODE  ## (1 line of code)
        mse = mean_squared_error(y_train[test], y_pred)
        avg_mse += mse
    ##  END CODE    ##

    # Take the average mean squared error as metric
    ridge_avg_mse[alpha] = avg_mse / n_folds


# Find the best value for alpha with minimum mean squared error
##  START CODE  ## (1 line of code)
best_alpha_ridge = min(ridge_avg_mse, key=ridge_avg_mse.get)
##  END CODE    ##

print("Best Ridge alpha: {}".format(best_alpha_ridge))
```

```
Best Ridge alpha: 57.223676593502205
```

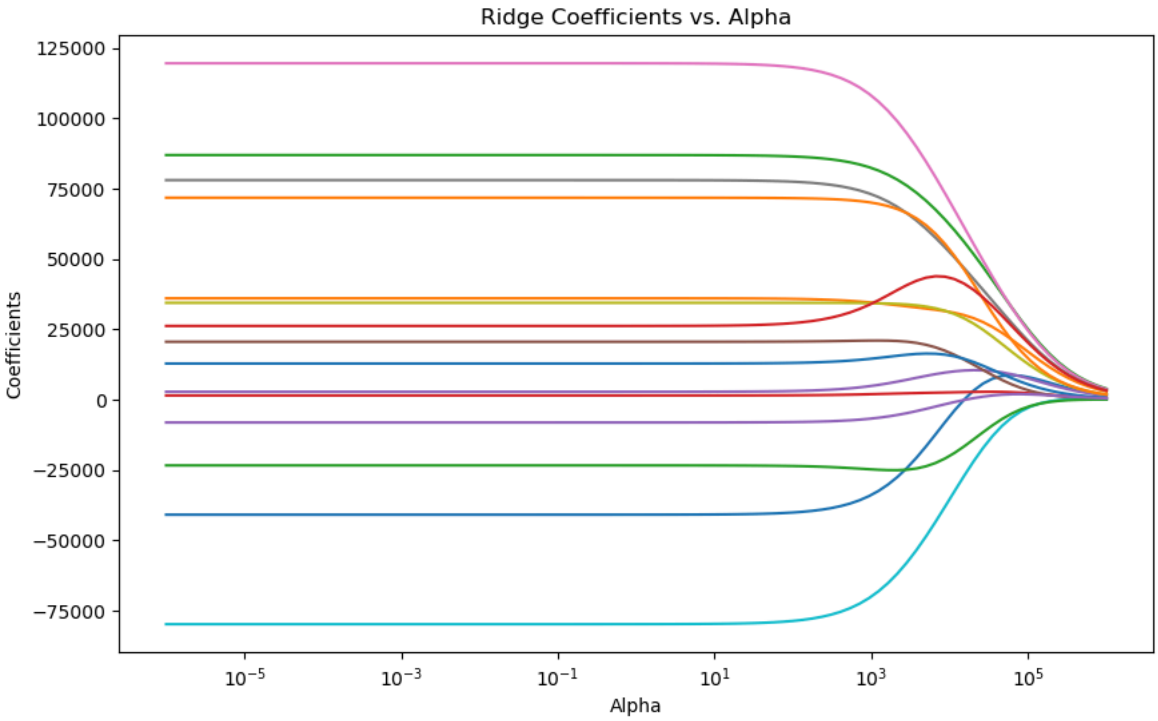
Part-3: (7 pts)

Run ridge and lasso regression for all of the α specified above (on training data), and plot the coefficients learned for each of them - there should be one plot each for lasso and ridge, so a total of two plots; different features' weights of each model should be on the same plot with different colors.

```
warnings.filterwarnings("ignore", category=ConvergenceWarning)

# Lasso Regression

alphas = 10**np.linspace(6,-6,100)

lasso = linear_model.Lasso(alpha=alpha)
coefs = []

for a in alphas:
  #Specify current alpha as parameter for the lasso model
  ## START CODE ## (1 line of code)
    lasso.set_params(alpha=a)
  ## END CODE ##

  #Fit the training data to the lasso model
  ## START CODE ## (1 line of code)
    lasso.fit(X_train_scaled, y_train)
  ## END CODE ##

  #Store learned coefficients in the coef variable
  ## START CODE ## (1 line of code)
    coefs.append(lasso.coef_)
  ## END CODE ##


# Write the code to make the plot for coefficients learned from lasso
## START CODE ##
plt.figure(figsize=(10, 6))
plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('Coefficients')
plt.title('Lasso Coefficients vs. Alpha')
plt.axis('tight')
plt.show()
## END CODE ##
```



```
# Ridge Regression

ridge = Ridge()
ridge_coefs = []

for a in alphas:
  #Specify current alpha as parameter for the lasso model
  ## START CODE ## (1 line of code)
    ridge.set_params(alpha=a)
  ## END CODE ##

  #Fit the training data to the lasso model
  ## START CODE ## (1 line of code)
    ridge.fit(X_train_scaled, y_train)

  ## END CODE ##

  #Store learned coefficients in the coef variable
  ## START CODE ## (1 line of code)
    ridge_coefs.append(ridge.coef_)
  ## END CODE ##


# Write the code to make the plot for coefficients learned from ridge
## START CODE ##
plt.figure(figsize=(10, 6))
plt.plot(alphas, ridge_coefs)
plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('Coefficients')
plt.title('Ridge Coefficients vs. Alpha')
plt.axis('tight')
plt.show()
## END CODE ##
```



What do you qualitatively observe when the value of the regularization parameter changes?

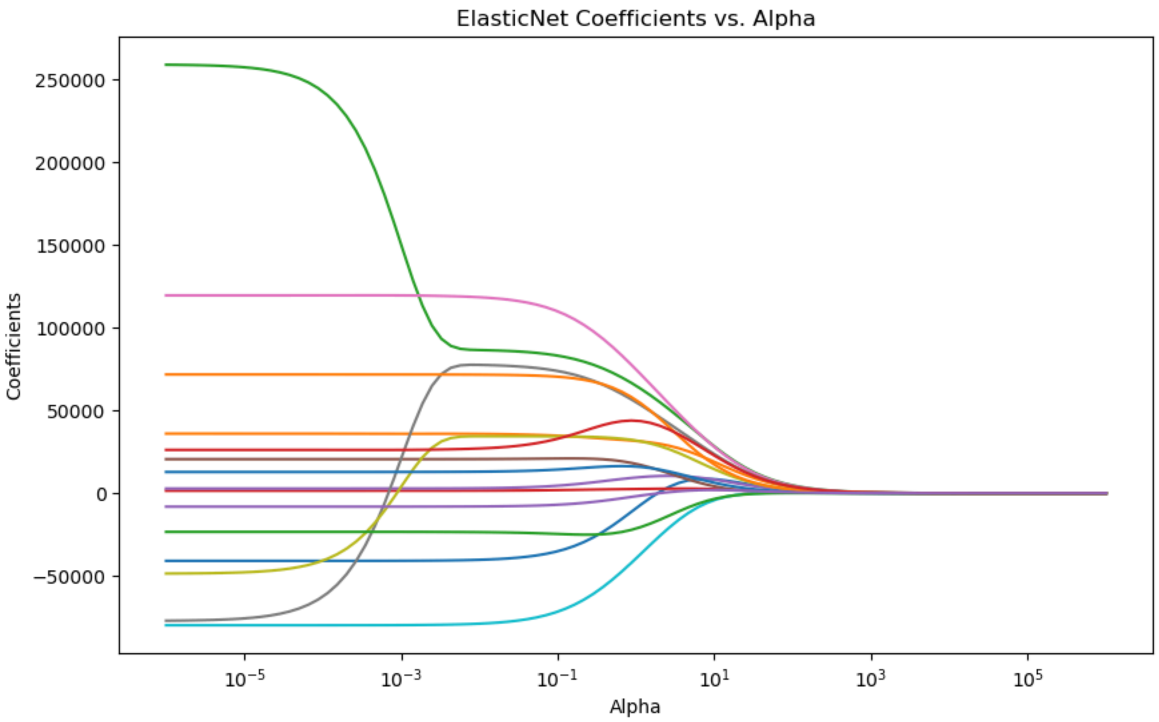
Answer

As the alpha value (regularization strength) increases in both Lasso and Ridge regression, the model penalizes larger coefficients by shrinking them toward zero. In Ridge regression, the coefficients are reduced gradually but never exactly to zero, retaining all features. In contrast, Lasso regression forces some coefficients to zero much more quickly, effectively removing irrelevant features and performing feature selection. Thus, Lasso tends to produce sparser models compared to Ridge

Part-4: (5 pts)

Similarly, use `sklearn.linear_model.ElasticNet` to do linear regression with different α values, and plot the coefficients learned for each of them

```
warnings.filterwarnings("ignore", category=ConvergenceWarning)
# Ridge Regression

alphas = 10**np.linspace(6,-6,100)

elast_net= linear_model.ElasticNet(alpha=alpha)
coefs = []

for a in alphas:
  #Specify current alpha as parameter for the lasso model
  ## START CODE ## (1 line of code)
    elast_net.set_params(alpha=a)
  ## END CODE ##

  #Fit the training data to the lasso model
  ## START CODE ## (1 line of code)
    elast_net.fit(X_train_scaled, y_train)
  ## END CODE ##

  #Store learned coefficients in the coef variable
  ## START CODE ## (1 line of code)
    coefs.append(elast_net.coef_)
  ## END CODE ##


# Write the code to make the plot for coefficients learned from ElasticNet
## START CODE ##
plt.figure(figsize=(10, 6))
plt.plot(alphas, coefs)
plt.xscale('log')
plt.xlabel('Alpha')
plt.ylabel('Coefficients')
plt.title('ElasticNet Coefficients vs. Alpha')
plt.axis('tight')
plt.show()
## END CODE ##
```



Observe the plot, then explain the pros and cons of ridge, lasso and Elastic Net models.

Answer

Ridge regression reduces model complexity by shrinking coefficients but doesn't eliminate any, making it suitable when all features are important but not for feature selection. Lasso, on the other hand, performs feature selection by driving some coefficients to zero, which simplifies the model but can struggle with correlated features. Elastic Net combines both approaches, offering a balance between feature selection and regularization, making it effective for dealing with correlated features. Ridge is more stable for multicollinearity, while Lasso produces sparse models.

Part-5: (10 pts)

Run the following three regression models with MSE loss on the training data:

(a) linear regression without regularization

(b) linear regression with ridge regularization

(c) linear regression with lasso regularization

For part (b) and (c), use only the best regularization parameters. Report the MSE and R2 on the test data for each of the models.

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error, r2_score
## START CODE ##
# (a) Linear regression without regularization
# Instantiate the linear regression model
lin_reg = LinearRegression()

# Fit the model on the training data
lin_reg.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred_lin = lin_reg.predict(X_test_scaled)
```

```
# Calculate MSE and R2
mse_lin = mean_squared_error(y_test, y_pred_lin)
r2_lin = r2_score(y_test, y_pred_lin)

print(f"Linear Regression MSE: {mse_lin:.4f}, R2: {r2_lin:.4f}")

# (b) Linear regression with Ridge regularization
# Instantiate the Ridge model with the best alpha
ridge_reg = Ridge(alpha=best_alpha_ridge)

# Fit the model on the training data
ridge_reg.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred_ridge = ridge_reg.predict(X_test_scaled)

# Calculate MSE and R2
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Ridge Regression MSE: {mse_ridge:.4f}, R2: {r2_ridge:.4f}")

# (c) Linear regression with Lasso regularization
# Instantiate the Lasso model with the best alpha
lasso_reg = Lasso(alpha=best_alpha_lasso)

# Fit the model on the training data
lasso_reg.fit(X_train_scaled, y_train)

# Predict on the test data
y_pred_lasso = lasso_reg.predict(X_test_scaled)

# Calculate MSE and R2
mse_lasso = mean_squared_error(y_test, y_pred_lasso)
r2_lasso = r2_score(y_test, y_pred_lasso)

print(f"Lasso Regression MSE: {mse_lasso:.4f}, R2: {r2_lasso:.4f}")

## END CODE ##
```

```
Linear Regression MSE: 48827883407.8462, R2: 0.6513
Ridge Regression MSE: 48828862496.5081, R2: 0.6512
Lasso Regression MSE: 48866301676.5811, R2: 0.6510
```

Part-6: (3 pts)

Train the 3 models and report metrics with the original data without scaling

```
## START CODE ##

# (a) Linear regression without regularization on original data
# Instantiate the linear regression model
lin_reg_orig = LinearRegression()

# Fit the model on the original training data
lin_reg_orig.fit(X_train, y_train)

# Predict on the original test data
y_pred_lin_orig = lin_reg_orig.predict(X_test)

# Calculate MSE and R2
mse_lin_orig = mean_squared_error(y_test, y_pred_lin_orig)
r2_lin_orig = r2_score(y_test, y_pred_lin_orig)

print(f"Linear Regression (No Scaling) MSE: {mse_lin_orig:.4f}, R2: {r2_lin_orig:.4f}")

# (b) Linear regression with Ridge regularization on original data
# Instantiate the Ridge model with the best alpha (previously obtained)
ridge_reg_orig = Ridge(alpha=best_alpha_ridge)

# Fit the model on the original training data
ridge_reg_orig.fit(X_train, y_train)

# Predict on the original test data
y_pred_ridge_orig = ridge_reg_orig.predict(X_test)

# Calculate MSE and R2
mse_ridge_orig = mean_squared_error(y_test, y_pred_ridge_orig)
r2_ridge_orig = r2_score(y_test, y_pred_ridge_orig)
```

```
print(f"Ridge Regression (No Scaling) MSE: {mse_ridge_orig:.4f}, R2: {r2_ridge_orig:.4f}")

# (c) Linear regression with Lasso regularization on original data
# Instantiate the Lasso model with the best alpha (previously obtained)
lasso_reg_orig = Lasso(alpha=best_alpha_lasso)

# Fit the model on the original training data
lasso_reg_orig.fit(X_train, y_train)

# Predict on the original test data
y_pred_lasso_orig = lasso_reg_orig.predict(X_test)

# Calculate MSE and R2
mse_lasso_orig = mean_squared_error(y_test, y_pred_lasso_orig)
r2_lasso_orig = r2_score(y_test, y_pred_lasso_orig)

print(f"Lasso Regression (No Scaling) MSE: {mse_lasso_orig:.4f}, R2: {r2_lasso_orig:.4f}")

## END CODE ##
```

```
Linear Regression (No Scaling) MSE: 48852207456.8430, R2: 0.6511
Ridge Regression (No Scaling) MSE: 49078487402.6339, R2: 0.6495
Lasso Regression (No Scaling) MSE: 48934697060.7969, R2: 0.6505
```

Part-7: (3 pts)

Why did we have to scale the data in ridge and lasso regression?

Answer

Scaling the data in Ridge and Lasso regression is crucial because it ensures that each feature contributes equally to the regularization penalty. Without scaling, features with larger ranges would disproportionately influence the model, leading to biased coefficient shrinkage. By standardizing the features, we ensure that the regularization affects all features uniformly.