HW 1

Shashank Rao- sr55952 , Sarthak shivnani - ss223347

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)

1) Amazon SageMaker is a fully managed machine learning platform that handles the operations part of integrating machine learning models into applications. It ensures high availability with multiple instances, eliminating downtime and maintenance windows. SageMaker offers flexibility by supporting a wide range of frameworks like TensorFlow, PyTorch, and MXNet, and allows users to create custom algorithms. Users can define their own pre- and post-processing steps and customize workflows at every stage of the ML pipeline to meet specific needs. By using SageMaker's machine learning capabilities, businesses can build ML models that deeply analyze data and extract meaningful insights for their operations. Some key benefits of Amazon SageMaker are:

Strong community and support: SageMaker has a robust community of data scientists and ML engineers, ensuring it stays up to date with the latest trends. Its presence, especially on GitHub, makes the code even more accessible to users. Additionally, SageMaker's integration with other AWS services like S3, EC2, and DynamoDB makes it even more beneficial for businesses to use, eliminating the need for dedicated manpower for each of these services.

2)Amazon SageMaker provides businesses with a comprehensive platform to streamline the development, training, and deployment of machine learning (ML) models. One of its standout features is its fully managed environment, which simplifies the operational complexities of running ML models. By integrating with other AWS services like S3, EC2, and DynamoDB, SageMaker offers seamless data management and compute resources, making it easier for businesses to leverage powerful ML models without needing dedicated infrastructure or personnel for each service. SageMaker's flexibility allows developers to use popular frameworks like TensorFlow and PyTorch, while also offering built-in algorithms tailored for various use cases such as classification, regression, and forecasting.

Additionally, SageMaker provides robust scalability and automation capabilities, helping businesses train and deploy models efficiently. Features like auto-scaling ensure that resources are optimized based on demand, while AutoML functionality simplifies model tuning and selection by automatically identifying the best algorithms for the data. SageMaker's integrated monitoring and debugging tools, like the SageMaker Debugger, enable developers to detect and resolve issues during training, making it a versatile and powerful platform for machine learning development across various industries.

3)Explanation 1 has a more casual and brief tone, Explanation 2 is more formal and comprehensive, covering multiple aspects of SageMaker's capabilities.Explanation 1 is narrower in scope, focusing on community support and AWS service integration as the primary benefits.Explanation 2 offers a much broader perspective, covering not just community support and integration but also SageMaker's full lifecycle capabilities. Explanation 1 provides fewer technical details, focusing more on the general benefits of integration and community support.Explanation 2 is more detailed technically. Explanation 1 seems targeted at a business or non-technical audience. Explanation 2 is aimed at a more technical audience

Overall, Explanation 1 is shorter, more focused on community and integration benefits, and is easier for a general audience to understand. Explanation 2 provides a more technical and comprehensive view of SageMaker's full capabilities, including automation, scalability, and tools for efficient ML model development and deployment, making it more suitable for a technically inclined audience.

Answer

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)

Part- 1 Two important causes of "data drift" in the Google Flu Trends (GFT) prediction problem mentioned in the article are:

1. Changes in Google's Search Algorithm: Google frequently updates its search algorithm to improve user experience and profitability. These changes, such as suggesting additional search terms, altered the data used by GFT making the flu-related search patterns less stable over time, leading to GFT's predictive errors.
2. Impact of Media: Media coverage caused people to search for flu-related information more often during flu season or outbreaks like the 2009 H1N1 pandemic. These search spikes didn't always match real flu cases, leading the GFT model to overestimate flu numbers.

Part 2

Transparency is crucial for building data science and AI projects for several reasons:

1. Reduces Errors and Misuse: Transparent communication across stakeholders ensures that AI models are designed, developed, and used properly. This prevents misunderstandings, such as optimizing for the wrong variable or incorrect handling of AI outputs that could lead to inefficiencies or even harm.
2. Enables Oversight: For internal risk management and external regulatory bodies, transparency is essential for overseeing AI projects. Clear communication about how the AI works and the decision behind its design helps identify risks, whether ethical, legal which allows for timely interventions.
3. Distributes Responsibility: Transparency allows for accountability to be shared among all involved parties, from executives to developers to end-users.
4. Respects Users: It allows users to make informed choices and prevents manipulation, thereby fostering trust between AI providers and users.

Answer

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.

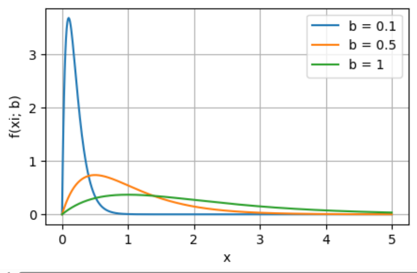
Suppose we observe 6 values obtained by sampling i.i.d. from the pdf described above:

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

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(xi; b)')
plt.legend()
plt.grid(True)
plt.show()
```



Answer

1. Log-Likelihood Expression

Given the probability density function (pdf):

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

we need to find the log-likelihood of observing a set of 6 samples $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ drawn i.i.d. from this distribution.

1. *Likelihood Function:*

   For a single observation x_i , the likelihood function is:

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

For 6 i.i.d. samples, the likelihood function (L(b)) is:

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

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

2. *Log-Likelihood Function:*

   Taking the logarithm to get the log-likelihood function $\ell(b)$:

$$\ell(b) = \log L(b)$$

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

2. Maximum Likelihood Estimate for (b)

To find the maximum likelihood estimate (MLE) for (b), we maximize the log-likelihood function $\ell(b)$.

1. *Differentiate the Log-Likelihood Function:*

$$\frac{\partial \ell(b)}{\partial b} = -12\frac{1}{b} + \frac{\sum_{i=1}^{6} x_i}{b^2}$$

2. *Set the Derivative to Zero:*

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

3. *Calculate the MLE:*

   For the sample values ({0.1, 0.5, 1, 0.4, 2, 1.4}):

$$\sum_{i=1}^{6} x_i = 0.1 + 0.5 + 1 + 0.4 + 2 + 1.4 = 5.4$$
$$b_{\text{MLE}} = \frac{5.4}{12} = 0.45$$

i.

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

ii. maximum likelihood estimate for the parameter b = 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-squared (R^2) and Adjusted R-squared are both measures used to assess the goodness-of-fit of a regression model, but they differ in how they account for the number of predictors in the model.

R-squared (R^2):

R^2 represents the proportion of the variance in the dependent variable that is explained by the independent variables in the model. It ranges from 0 to 1, where a higher value indicates a better fit. A key drawback of R^2 is that it always increases or remains the same as more independent variables are added to the model, even if those variables do not improve the model's predictive power. This can lead to overfitting. Adjusted R-squared:

Adjusted R^2 adjusts the R^2 value based on the number of predictors in the model and the sample size.

It penalizes the addition of irrelevant predictors that do not improve the model.

Unlike R^2, adjusted R^2 can decrease if the new predictors don't improve the model fit.

2. Why we use Adjusted R^2? Adjusted R^2 is desirable because it provides a more accurate measure of how well the model explains the data by accounting for the number of predictors. It helps prevent overfitting, where R^2 may artificially inflate due to the addition of unnecessary variables. Adjusted R^2 is more reliable for comparing models with different numbers of predictors.

Double-click (or enter) to edit

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()
```

```
Choose Files No file chosen          Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving house_cost_data.csv to house_cost_data.csv
```

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_year	renovation_year	latitude	longitude	sqft_living15	sqft_lot15
0	221900.0	3	1.00	1180	5650	1.0	3	7	1180	0	1955	0	47.5112	-122.257	1340	5650
1	538000.0	3	2.25	2570	7242	2.0	3	7	2170	400	1951	1991	47.7210	-122.319	1690	7639
2	180000.0	2	1.00	770	10000	1.0	3	6	770	0	1933	0	47.7379	-122.233	2720	8062
3	604000.0	4	3.00	1960	5000	1.0	5	7	1050	910	1965	0	47.5208	-122.393	1360	5000
4	510000.0	3	2.00	1680	8080	1.0	3	8	1680	0	1987	0	47.6168	-122.045	1800	7503

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)
```

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 StandardScaler

# Create a StandardScaler object
scaler = StandardScaler()

# Fit and transform the training data
Xscaled_train = scaler.fit_transform(X_train)

# Transform the test data using the same scaler
Xscaled_test = scaler.transform(X_test)
```

Print the first 5 rows of the training set after scaling

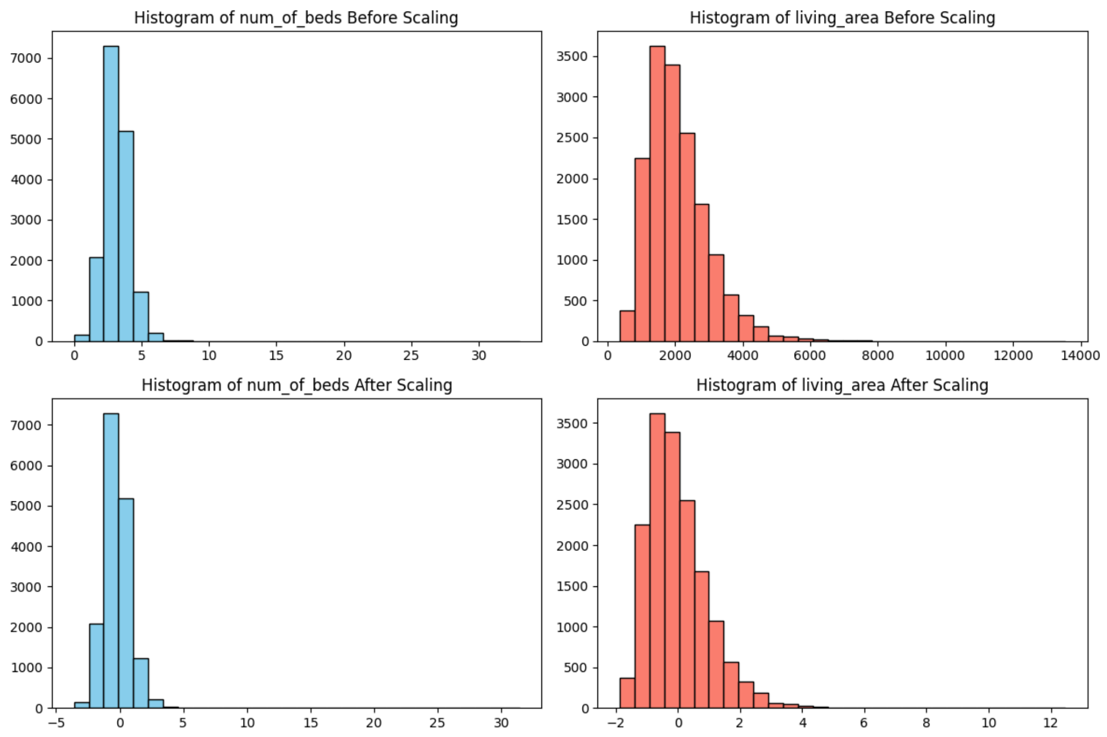
```
##  START CODE  ##
Xscaled_train[:5]
##  END CODE    ##
```

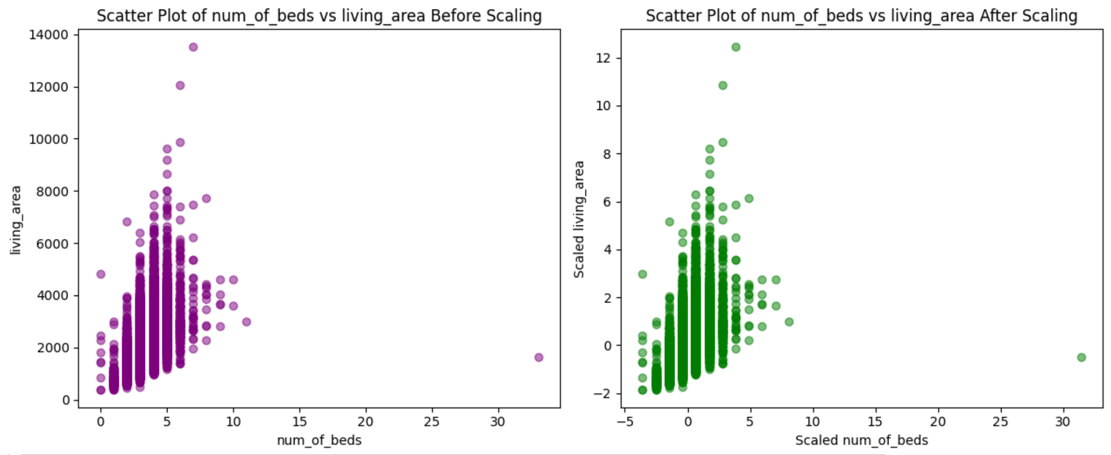
```
array([[-1.46128799, -1.45010618, -1.36293279, -0.22520747, -0.92013997,
         0.90694939, -1.41249264, -1.16171819, -0.65460814, -0.58245852,
        -0.20780192,  1.34578638, -0.85717047, -0.94773519, -0.24597874],
       [-1.46128799, -1.45010618, -1.37381314,  0.02822559, -0.92013997,
         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  ##
import matplotlib.pyplot as plt

var1, var2 = 'num_of_beds', 'living_area'

plt.figure(figsize=(12, 8))

# Histogram of var1 before scaling
plt.subplot(2, 2, 1)
plt.hist(X_train[var1], bins=30, color='skyblue', edgecolor='black')
plt.title(f'Histogram of {var1} Before Scaling')
```

```
# Histogram of var2 before scaling
plt.subplot(2, 2, 2)
plt.hist(X_train[var2], bins=30, color='salmon', edgecolor='black')
plt.title(f'Histogram of {var2} Before Scaling')

# Histogram of var1 after scaling
plt.subplot(2, 2, 3)
plt.hist(Xscaled_train[:, X_train.columns.get_loc(var1)], bins=30, color='skyblue', edgecolor='black')
plt.title(f'Histogram of {var1} After Scaling')

# Histogram of var2 after scaling
plt.subplot(2, 2, 4)
plt.hist(Xscaled_train[:, X_train.columns.get_loc(var2)], bins=30, color='salmon', edgecolor='black')
plt.title(f'Histogram of {var2} After Scaling')

plt.tight_layout()
plt.show()

# Scatter plot before and after scaling
plt.figure(figsize=(12, 5))

# Scatter plot before scaling
plt.subplot(1, 2, 1)
plt.scatter(X_train[var1], X_train[var2], color='purple', alpha=0.5)
plt.title(f'Scatter Plot of {var1} vs {var2} Before Scaling')
plt.xlabel(var1)
plt.ylabel(var2)

# Scatter plot after scaling
plt.subplot(1, 2, 2)
plt.scatter(Xscaled_train[:, X_train.columns.get_loc(var1)], Xscaled_train[:, X_train.columns.get_loc(var2)], color='green', alpha=0.5)
plt.title(f'Scatter Plot of {var1} vs {var2} After Scaling')
plt.xlabel(f'Scaled {var1}')
plt.ylabel(f'Scaled {var2}')

plt.tight_layout()
plt.show()
##  END CODE    ##
```





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)
##  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 = {}


#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 = linear_model.Lasso(alpha=alpha)
  ##  END CODE    ##

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

    #Fit the scaled training data to the lasso model
    ## START CODE ## (1 line of code)
    lasso.fit(Xscaled_train[train], y_train.iloc[train])
    ## END CODE ##

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

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

Show hidden output

```
# 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: 403.70172585965497
```

```
#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 = linear_model.Ridge(alpha=alpha)
  ##  END CODE    ##

  avg_mse = 0

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

    #Fit the scaled training data to the ridge model
    ## START CODE ## (1 line of code)
    ridge.fit(Xscaled_train[train], y_train.iloc[train])
    ## END CODE ##

    #Calculate the average mean sqaured error
    ##  START CODE  ## (1 line of code)
    avg_mse += mean_squared_error(y_train.iloc[test], ridge.predict(Xscaled_train[test]))
    ##  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(lasso_avg_mse, key=ridge_avg_mse.get)
##  END CODE    ##

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

```
Best Ridge alpha: 100.0
```

Part-3: (7 pts)

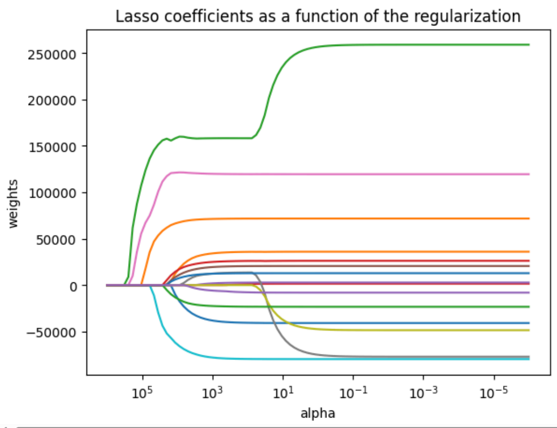
Run ridge and lasso regression for all of the α specified above (on training data), and plot the coefficients learned for each of them - there should be one plot each for lasso and ridge, so a total of two plots; different features' weights of each model should be on the same plot with different colors.

```
# Lasso Regression
```

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

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

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(Xscaled_train, y_train)
  ## END CODE ##

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

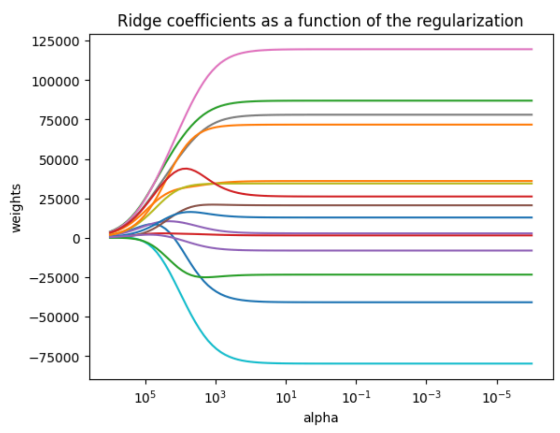
Show hidden output

```
# Write the code to make the plot for coefficients learned from lasso
## START CODE ##
import matplotlib.pyplot as plt

ax = plt.gca()

ax.plot(alphas, coefs_lasso)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization')
plt.axis('tight')
plt.show()
## END CODE ##
```



```
# Ridge Regression

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

ridge = linear_model.Ridge(alpha=alpha)
coefs_ridge = []

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(Xscaled_train, y_train)
  ## END CODE ##

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

```
# Write the code to make the plot for coefficients learned from ridge
## START CODE ##
ax = plt.gca()

ax.plot(alphas, coefs_ridge)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()
## END CODE ##
```



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

Answer

When the regularization parameter alpha increases in Lasso regression, coefficients are aggressively pushed to zero, eliminating less important features and simplifying the model. As alpha decreases, more features are retained, increasing the risk of overfitting.

In Ridge regression, increasing alpha shrinks all coefficients smoothly toward zero but never eliminates them, resulting in a simpler model without excluding features. Decreasing alpha increases the coefficients, making the model more complex and prone to overfitting.

Lasso eliminates features entirely, while Ridge reduces their impact without excluding any.

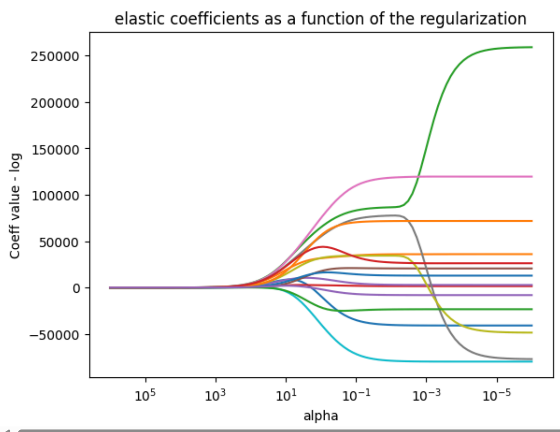
Part-4: (5 pts)

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

```
from sklearn.linear_model import ElasticNet
# Ridge Regression

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

ElastNet = linear_model.ElasticNet(alpha=alpha)
coefs_elastic = []

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

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

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

Show hidden output

```
# Write the code to make the plot for coefficients learned from ElasticNet
## START CODE ##
ax = plt.gca()

ax.plot(alphas, coefs_elastic)
ax.set_xscale('log')
ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
plt.xlabel('alpha')
plt.ylabel('Coeff value - log')
plt.title('elastic coefficients as a function of the regularization')
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:

- **Pros:** Reduces model complexity by shrinking coefficients, helps prevent overfitting, works well with many small/medium effects, and handles multicollinearity.
- **Cons:** Does not perform feature selection, leading to less interpretable models since all features are retained.

Lasso Regression:

- **Pros:** Performs feature selection by driving some coefficients to zero, making models simpler and more interpretable, useful when only a few features are important.
- **Cons:** Sensitive to data changes, especially when features are highly correlated, and may overlook relevant features when many contribute similarly.

Elastic Net:

- **Pros:** Combines Ridge and Lasso strengths with both L1 and L2 penalties, effective for high-dimensional datasets, and handles correlated features well.
- **Cons:** More complex due to two hyperparameters, requires careful tuning for optimal performance.

Elastic Net is often preferred for high-dimensional data, balancing the trade-offs between Ridge and Lasso.

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.

```
!pip install scikit-learn
import numpy as np
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score

## START CODE ##
# (a) Linear Regression without regularization
linear_reg = LinearRegression()
linear_reg.fit(Xscaled_train, y_train)
y_pred_linear = linear_reg.predict(X_test)

# (b) Linear Regression with Ridge regularization
ridge = Ridge()
parameters = {'alpha': 10**np.linspace(10, -2, 100) * 0.5}
ridge_reg = GridSearchCV(ridge, parameters, scoring='neg_mean_squared_error', cv=5)
ridge_reg.fit(Xscaled_train, y_train)
y_pred_ridge = ridge_reg.predict(X_test)

# (c) Linear Regression with Lasso regularization
lasso = Lasso()
parameters = {'alpha': 10**np.linspace(10, -2, 100) * 0.5}
lasso_reg = GridSearchCV(lasso, parameters, scoring='neg_mean_squared_error', cv=5)
lasso_reg.fit(Xscaled_train, y_train)
y_pred_lasso = lasso_reg.predict(X_test)

# Calculate MSE and R2 for each model
models = [linear_reg, ridge_reg, lasso_reg]
model_names = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']

for model, name in zip(models, model_names):
  print(f'Model: {name}')
  print(f'MSE: {mean_squared_error(y_test, model.predict(Xscaled_test))}')
  print(f'R2: {r2_score(y_test, model.predict(Xscaled_test))}')
  print('-' * 20)
## END CODE ##
```

Show hidden output

```
for model, name in zip(models, model_names):
  print(f'Model: {name}')
  print(f'MSE: {mean_squared_error(y_test, model.predict(Xscaled_test))}')
  print(f'R2: {r2_score(y_test, model.predict(Xscaled_test))}')
  print('-' * 20)
```

```
Model: Linear Regression
MSE: 1666157980015922.2
R2: -11899.19357588803
--------------------
Model: Ridge Regression
MSE: 1651381511160447.5
R2: -11793.6556606019
--------------------
Model: Lasso Regression
MSE: 1606167571616564.0
R2: -11470.724318343724
--------------------
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but Ridge was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but Ridge was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but Lasso was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465: UserWarning: X does not have valid feature names, but Lasso was fitted with feature names
  warnings.warn(
```

Part-6: (3 pts)

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

```
## START CODE ##
# (a) Linear Regression without regularization
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
y_pred_linear = linear_reg.predict(X_test)

# (b) Linear Regression with Ridge regularization
ridge = Ridge()
parameters = {'alpha': 10**np.linspace(10, -2, 100) * 0.5}
ridge_reg = GridSearchCV(ridge, parameters, scoring='neg_mean_squared_error', cv=5)
ridge_reg.fit(X_train, y_train)
y_pred_ridge = ridge_reg.predict(X_test)

# (c) Linear Regression with Lasso regularization
lasso = Lasso()
parameters = {'alpha': 10**np.linspace(10, -2, 100) * 0.5}
lasso_reg = GridSearchCV(lasso, parameters, scoring='neg_mean_squared_error', cv=5)
lasso_reg.fit(X_train, y_train)
y_pred_lasso = lasso_reg.predict(X_test)

# Calculate MSE and R2 for each model
models = [linear_reg, ridge_reg, lasso_reg]
model_names = ['Linear Regression', 'Ridge Regression', 'Lasso Regression']

for model, name in zip(models, model_names):
  print(f'Model: {name}')
  print(f'MSE: {mean_squared_error(y_test, model.predict(X_test))}')
  print(f'R2: {r2_score(y_test, model.predict(X_test))}')
  print('-' * 20)
## END CODE ##
```

Show hidden output

```
for model, name in zip(models, model_names):
  print(f'Model: {name}')
  print(f'MSE: {mean_squared_error(y_test, model.predict(X_test))}')
  print(f'R2: {r2_score(y_test, model.predict(X_test))}')
  print('-' * 20)
```

```
Model: Linear Regression
MSE: 48852207456.84297
R2: 0.6510830711621256
--------------------
Model: Ridge Regression
MSE: 48853953233.84231
R2: 0.6510706023059418
--------------------
Model: Lasso Regression
MSE: 48858511085.050255
R2: 0.6510380487832175
--------------------
```

Part-7: (3 pts)

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

Answer

Scaling is necessary in Ridge and Lasso regression because the regularization penalties are sensitive to feature magnitudes. Without scaling, features with larger values could dominate the model, leading to biased coefficients. Scaling ensures that all features contribute equally and regularization is applied fairly.

Start coding or generate with AI.