Capstone Example

Last week's notebook discussed a cummulative project that would be used as a test of knowledge from this series of courses. This notebook will serve as a reference point for you while you work on said project. Included in this notebook is a set of answers to your tasks, based on a set dataset. Make sure in your final submission you are using a different dataset!

You will be working on 4 tasks:

1. **Data Processing**
2. **Classification**
3. **Regression**
4. **Recommender Sytstems**

These tasks are each representative of one of the courses in the series. So if you need help with any one of these tasks, be sure to look back at those courses for reference! Along with the previous courses, there will be checkpoints with given solutions so you can check to make sure you are headed in the right direction. ***Good Luck!***

Dataset Description

The dataset analyzed in this project is titled "Amazon Musical Instruments Reviews." It comprises 904,004 rows and 15 columns, providing detailed information about customer reviews for products listed on Amazon. A comprehensive description of each column is provided below:

1. **marketplace:** The region or country (e.g., US) where the review was submitted.
2. **customer_id:** A unique identifier for the customer who submitted the review.
3. **review_id:** A unique identifier for the review.
4. **product_id:** A unique identifier for the product.
5. **product_parent:** An internal Amazon product identifier, used to group products under a parent category or product line.
6. **product_title:** The title or name of the product, providing a description of the item.
7. **product_category:** The category of the product, such as "Musical Instruments" in this dataset.
8. **star_rating:** The star rating given by the user to the product, ranging from 1 to 5 stars.
9. **helpful_votes:** The number of votes the review received as helpful from other users.
10. **total_votes:** The total number of votes the review received, including both helpful and not helpful votes.
11. **vine:** Indicates whether the review was part of Amazon's Vine Program.
12. **verified_purchase:** Indicates whether the review came from a verified purchase.
13. **review_headline:** The headline of the review, summarizing its main points.
14. **review_body:** The full textual content of the review, describing the user's experience and feedback about the product.
15. **review_date:** The date the review was submitted.

Task 1: Data Processing

The Data

For this final project you will be doing your work on a dataset of your choice. For reference, an example with checkpoint answers will be included. This example will be an amazon dataset, which does not need any cleaning before proper analysis. This dataset in particular can be found [here. (https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Home_Improvement_v1_00.tsv.gz)](https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Home_Improvement_v1_00.tsv.gz) This dataset is a set of Home Improvement Product reviews on amazon. It is a rather large dataset, so our computation might take slightly longer than normal.

First Step: Imports

In the next cell we will give you all of the imports you should need to do your project. Feel free to add more if you would like, but these should be sufficient.

In [1]:
```
import gzip
from collections import defaultdict
import random
import numpy as np
import numpy
import scipy.optimize
import string
from sklearn import linear_model
from nltk.stem.porter import PorterStemmer # Stemming
import pandas as pd
import zipfile
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
from sklearn.feature_extraction.text import CountVectorizer
from scipy.optimize import fmin_l_bfgs_b
```

TODO 1: Read the data and Fill your dataset

Take care of int casting the votes and rating. Also **add this bit of code** to your for loop, taking off the outer " ":

" d['verified_purchase'] = d['verified_purchase'] == 'Y' "

This simple makes the verified purchase column be strictly true/false values rather than Y/N strings.

In [4]:
```
# Path to the uploaded file
file_path = 'amazon_reviews_us_Musical_Instruments_v1_00.tsv'

# Extract and load the complete data
df = pd.read_csv(file_path, sep='\t', on_bad_lines='skip')

# Display dataset info and the first few rows
print(df.info())  # Summary of the dataset
print(df.head())  # Display the first few rows
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904004 entries, 0 to 904003
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   marketplace        904004 non-null  object
 1   customer_id        904004 non-null  int64 
 2   review_id          904004 non-null  object
 3   product_id         904004 non-null  object
 4   product_parent     904004 non-null  int64 
 5   product_title      904003 non-null  object
 6   product_category   904004 non-null  object
 7   star_rating        904004 non-null  int64 
 8   helpful_votes      904004 non-null  int64 
 9   total_votes        904004 non-null  int64 
 10  vine               904004 non-null  object
 11  verified_purchase  904004 non-null  object
 12  review_headline    903998 non-null  object
 13  review_body        903941 non-null  object
 14  review_date        903996 non-null  object
dtypes: int64(5), object(10)
memory usage: 103.5+ MB
None
  marketplace  customer_id       review_id  product_id  product_parent  \
0          US     45610553   RMDCHWD0Y5OZ9  B00HH62VB6       618218723   
1          US     14640079   RZSL0BALIYUNU  B003LRN53I       986692292   
2          US      6111003   RIZR67JKUDBI0  B0006VMBHI       603261968   
3          US      1546619  R27HL570VNL85F  B002B55TRG       575084461   
4          US     12222213  R34EBU9QDWJ1GD  B00N1YPXW2       165236328   

                                       product_title     product_category  \
0  AGPtek® 10 Isolated Output 9V 12V 18V Guitar P...  Musical Instruments   
1        Sennheiser HD203 Closed-Back DJ Headphones  Musical Instruments   
2                   AudioQuest LP record clean brush  Musical Instruments   
3     Hohner Inc. 560BX-BF Special Twenty Harmonica  Musical Instruments   
4       Blue Yeti USB Microphone - Blackout Edition  Musical Instruments   

   star_rating  helpful_votes  total_votes vine verified_purchase  \
0            3              0            1    N                 N   
1            5              0            0    N                 Y   
2            3              0            1    N                 Y   
3            5              0            0    N                 Y   
4            5              0            0    N                 Y   

                                     review_headline  \
0                                        Three Stars   
1                                         Five Stars   
2                                        Three Stars   
3  I purchase these for a friend in return for pl...   
4                                         Five Stars   

                                         review_body review_date  
0        Works very good, but induces ALOT of noise.  2015-08-31  
1              Nice headphones at a reasonable price.  2015-08-31  
2                         removes dust. does not clean  2015-08-31  
3  I purchase these for a friend in return for pl...  2015-08-31  
4                              This is an awesome mic!  2015-08-31  
```

In [6]:
```
df['verified_purchase'] = df['verified_purchase'] == 'Y'
```

In [8]:
```
# Check for missing values before dropping
print("Missing values per column before dropping:")
print(df.isnull().sum())

# Drop rows with missing values
df_cleaned = df.dropna()

# Check for missing values after dropping
print("\nMissing values per column after dropping:")
print(df_cleaned.isnull().sum())

# Display the cleaned dataset summary
print("\nCleaned dataset info:")
print(df_cleaned.info())
```

```
Missing values per column before dropping:
marketplace           0
customer_id           0
review_id             0
product_id            0
product_parent        0
product_title         1
product_category      0
star_rating           0
helpful_votes         0
total_votes           0
vine                  0
verified_purchase     0
review_headline       6
review_body          63
review_date           8
dtype: int64

Missing values per column after dropping:
marketplace          0
customer_id          0
review_id            0
product_id           0
product_parent       0
product_title        0
product_category     0
star_rating          0
helpful_votes        0
total_votes          0
vine                 0
verified_purchase    0
review_headline      0
review_body          0
review_date          0
dtype: int64

Cleaned dataset info:
<class 'pandas.core.frame.DataFrame'>
Index: 903926 entries, 0 to 904003
Data columns (total 15 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   marketplace        903926 non-null  object
 1   customer_id        903926 non-null  int64 
 2   review_id          903926 non-null  object
 3   product_id         903926 non-null  object
 4   product_parent     903926 non-null  int64 
 5   product_title      903926 non-null  object
 6   product_category   903926 non-null  object
 7   star_rating        903926 non-null  int64 
 8   helpful_votes      903926 non-null  int64 
 9   total_votes        903926 non-null  int64 
 10  vine               903926 non-null  object
 11  verified_purchase  903926 non-null  bool  
 12  review_headline    903926 non-null  object
 13  review_body        903926 non-null  object
 14  review_date        903926 non-null  object
dtypes: bool(1), int64(5), object(9)
memory usage: 104.3+ MB
None
```

To do this setup properly, you **should** shuffle your data (which you should do in your submission), but the checkpoint values would change so for the sake of this example we will ***not*** shuffle the data.

TODO 2: Split the data into a Training and Testing set

Have Training be the first 80%, and testing be the remaining 20%.

In [11]:
```
def split_data_randomly(data, train_ratio=0.8, random_state=None):
    """
    Randomly splits the dataset into training and testing sets.
    :param data: The original dataset (DataFrame).
    :param train_ratio: The ratio of the data to be used as the training se
t (default is 80%).
    :param random_state: The seed for random number generation (optional, e
nsures reproducibility).
    :return: Two DataFrames - the training set and the testing set.
    """
    # Use train_test_split to split the dataset based on the specified trai
n/test ratio
    train_data, test_data = train_test_split(
        data, test_size=(1 - train_ratio), random_state=random_state
    )
    return train_data, test_data

train_data, test_data = split_data_randomly(df_cleaned, train_ratio=0.8, ra
ndom_state=42)


print(len(train_data), len(test_data))
```

```
723140 180786
```

Now delete your dataset

You don't want any of your answers to come from your original dataset any longer, but rather your Training Set, this will help you to not make any mistakes later on, especialy when referencing the checkpoint solutions.

In [14]:
```
del df
del df_cleaned
```

TODO 3: Extracting Basic Statistics

Next you need to answer some questions through any means (i.e. write a function or just find the answer) all based on the **Training Set:**

1. What is the **average rating**?
2. What fraction of reviews are from **verified purchases**?
3. How many **total users** are there?
4. How many **total items** are there?
5. What fraction of reviews have **5-star ratings**?

In [17]:
```
# 1. Function to calculate the average star rating
def calculate_average_rating(data):
    """
    Calculate the average star rating from the dataset.
    :param data: The dataset (DataFrame) containing the 'star_rating' colum
n.
    :return: The mean of the 'star_rating' column.
    """
    return data['star_rating'].mean()

# 2. Function to calculate the fraction of verified purchases
def calculate_verified_fraction(data):
    """
    Calculate the fraction of reviews that are verified purchases.
    :param data: The dataset (DataFrame) containing the 'verified_purchase'
column.
    :return: The mean value of the 'verified_purchase' column (True/False a
s 1/0).
    """
    return data["verified_purchase"].mean()

# 3. Function to calculate the total number of unique users
def calculate_total_users(data):
    """
    Calculate the total number of unique users who submitted reviews.
    :param data: The dataset (DataFrame) containing the 'customer_id' colum
n.
    :return: The count of unique customer IDs.
    """
    return data['customer_id'].nunique()

# 4. Function to calculate the total number of unique products
def calculate_total_items(data):
    """
    Calculate the total number of unique products reviewed in the dataset.
    :param data: The dataset (DataFrame) containing the 'product_id' colum
n.
    :return: The count of unique product IDs.
    """
    return data['product_id'].nunique()

# 5. Function to calculate the fraction of 5-star ratings
def calculate_five_star_fraction(data):
    """
    Calculate the fraction of reviews that have a 5-star rating.
    :param data: The dataset (DataFrame) containing the 'star_rating' colum
n.
    :return: The proportion of 5-star ratings in the dataset.
    """
    return (data['star_rating'] == 5).sum() / len(data)

# Call each function to compute metrics based on the training dataset
average_rating = calculate_average_rating(train_data)
verified_fraction = calculate_verified_fraction(train_data)
total_users = calculate_total_users(train_data)
total_items = calculate_total_items(train_data)
five_star_fraction = calculate_five_star_fraction(train_data)

# Display the results in a formatted output
print(f"1. Average Rating: {average_rating:.2f}")
```

```
print(f"2. Fraction of Verified Purchases: {verified_fraction:.2%}")
print(f"3. Total Users: {total_users}")
print(f"4. Total Items: {total_items}")
print(f"5. Fraction of 5-Star Ratings: {five_star_fraction:.2%}")
```

```
1. Average Rating: 4.25
2. Fraction of Verified Purchases: 86.38%
3. Total Users: 481417
4. Total Items: 111099
5. Fraction of 5-Star Ratings: 63.32%
```

Task 2: Classification

Next you will use our knowledge of classification to extract features and make predictions based on them. Here you will be using a Logistic Regression Model, keep this in mind so you know where to get help from.

TODO 1: Define the feature function

This implementation will be based on the **star rating** and the ***length*** of the **review body**. Hint: Remember the offset!

In [20]:
```
def create_features(data):
    """
    Define the feature extraction function based on review body length and star rating.
    :param data: The original dataset (DataFrame).
    :return: A feature matrix X and a target array y.
    """
    # Calculate the length of the review body
    review_length = data['review_body'].fillna('').apply(len).values
    # Extract the star ratings as features
    star_rating = data['star_rating'].values
    # Combine the features into a single feature matrix
    X = np.vstack((review_length, star_rating)).T
    # Extract the target variable (whether the purchase is verified or not)
    y = data['verified_purchase'].astype(int).values
    return X, y

# Generate features and target variables for the training dataset
X_train, y_train = create_features(train_data)
# Generate features and target variables for the testing dataset
X_test, y_test = create_features(test_data)
```

TODO 2: Fit your model

1. Create your **Feature Vector** based on your feature function defined above.
2. Create your **Label Vector** based on the "verified purchase" column of your training set.
3. Define your model as a **Logistic Regression** model.
4. Fit your model.

In [23]:
```
# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)
```

TODO 3: Compute Accuracy of Your Model

1. Make **Predictions** based on your model.
2. Compute the **Accuracy** of your model.

In [26]:
```
# Compute the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2%}")
```

```
Model Accuracy: 86.37%
```

TODO 4: Finding the Balanced Error Rate

1. Compute **True** and **False Positives**
2. Compute **True** and **False Negatives**
3. Compute **Balanced Error Rate** based on your above defined variables.

In [29]:
```
def compute_ber(y_true, y_pred):
    """
    Compute the Balanced Error Rate (BER).
    :param y_true: The true labels (ground truth).
    :param y_pred: The predicted labels.
    :return: The Balanced Error Rate (BER).
    """
    # Compute the confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Extract True Negatives (tn), False Positives (fp), False Negatives (f
n), and True Positives (tp) from the confusion matrix
    tn, fp, fn, tp = cm.ravel()
    # Calculate sensitivity (recall for the positive class)
    sensitivity = tp / (tp + fn)
    # Calculate specificity (recall for the negative class)
    specificity = tn / (tn + fp)
    # Compute the Balanced Error Rate
    ber = 1 - 0.5 * (sensitivity + specificity)
    return ber

# Calculate the Balanced Error Rate for the test dataset
ber = compute_ber(y_test, y_pred)
print(f"Balanced Error Rate (BER): {ber:.2%}")
```

```
Balanced Error Rate (BER): 48.12%
```

Task 3: Regression

In this section you will start by working though two examples of altering features to further differentiate. Then you will work through how to evaluate a Regularaized model.

Lets start by defining a new y vector, specific to our Regression model.

In [32]:
```
y_reg = train_data['star_rating'].values
```

TODO 1: Unique Words in a Sample Set

We are going to work with a smaller Sample Set here, as stemming on the normal training set will take a very long time. (Feel free to change sampleSet -> trainingSet if you would like to see)

1. Count the number of unique words found within the 'review body' portion of the sample set defined below, making sure to **Ignore Punctuation and Capitalization**.
2. Count the number of unique words found within the 'review body' portion of the sample set defined below, this time with use of **Stemming, Ignoring Puctuation, *and* Capitalization**.

In [35]:
```
#GIVEN for 1.
wordCount = defaultdict(int)
punctuation = set(string.punctuation)

#GIVEN for 2.
wordCountStem = defaultdict(int)
stemmer = PorterStemmer() #use stemmer.stem(stuff)
```

In [37]:
```
sampleSet = train_data[:2*len(train_data)//10]
```

In [39]:
```
# Count unique words without stemming
for review in sampleSet['review_body'].fillna(''):  # Fill missing values w
ith an empty string
    # Remove punctuation and convert to lowercase
    words = ''.join([char.lower() if char not in punctuation else ' ' for c
har in review]).split()
    # Count occurrences of each word
    for word in words:
        wordCount[word] += 1

# Count unique words with stemming
for review in sampleSet['review_body'].fillna(''):  # Fill missing values w
ith an empty string
    # Remove punctuation and convert to lowercase
    words = ''.join([char.lower() if char not in punctuation else ' ' for c
har in review]).split()
    # Apply stemming and count occurrences
    for word in words:
        stemmed_word = stemmer.stem(word)  # Apply stemming to the word
        wordCountStem[stemmed_word] += 1

# Calculate the number of unique words in both cases
unique_words_no_stemming = len(wordCount)
unique_words_with_stemming = len(wordCountStem)

# Print the results
print(f"Unique Words (No Stemming): {unique_words_no_stemming}")
print(f"Unique Words (With Stemming): {unique_words_with_stemming}")
```

```
Unique Words (No Stemming): 72285
Unique Words (With Stemming): 53648
```

TODO 2: Evaluating Classifiers

1. Given the feature function and your counts vector, **Define** your X_reg vector. (This being the X vector, simply labeled for the Regression model)
2. **Fit** your model using a **Ridge Model** with (alpha = 1.0, fit_intercept = True).
3. Using your model, **Make your Predictions**.
4. Find the **MSE** between your predictions and your y_reg vector.

In [41]:
```
#GIVEN FUNCTIONS
def feature_reg(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review_body'].lower() if not c in punctu
ation])
    for w in r.split():
        if w in wordSet:
            feat[wordId[w]] += 1
    return feat

def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [42]:
```
#GIVEN COUNTS AND SETS
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

#Note: increasing the size of the dictionary may require a lot of memory
words = [x[1] for x in counts[:100]]

wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

In [43]:
```
# Define the feature extraction function (based on word frequency)
def create_X_y_for_regression(data):
    """
    Create the feature matrix X and target vector y for regression.
    :param data: The original dataset (DataFrame).
    :return: Feature matrix X and target vector y.
    """
    # Use CountVectorizer to convert the text data into a bag-of-words repr
esentation
    # Only the top 100 most frequent words are selected as features
    vectorizer = CountVectorizer(max_features=100)

    # Transform the 'review_body' column into a feature matrix
    # Fill missing values with an empty string before transformation
    X = vectorizer.fit_transform(data['review_body'].fillna('')).toarray()

    # Extract the target variable (star ratings)
    y = data['star_rating'].values
    return X, y

# Create the feature matrix X and target vector y for regression
X_reg, y_reg = create_X_y_for_regression(train_data)

# Initialize the Ridge Regression model
ridge_model = Ridge(alpha=1.0, fit_intercept=True)

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_reg, y_reg, test_size=
0.2, random_state=42)

# Train the Ridge Regression model on the training data
ridge_model.fit(X_train, y_train)

# Predict the target variable for the validation set
y_pred = ridge_model.predict(X_val)

# Calculate the Mean Squared Error (MSE) to evaluate model performance
mse = mean_squared_error(y_val, y_pred)
print(f"Mean Squared Error (MSE): {mse:.2f}")
```

```
Mean Squared Error (MSE): 1.23
```

In [44]:
```
# If you would like to work with this example more in your free time, here
are some tips to improve your solution:
# 1. Implement a validation pipeline and tune the regularization parameter
# 2. Alter the word features (e.g. dictionary size, punctuation, capitaliza
tion, stemming, etc.)
# 3. Incorporate features other than word features
```

Task 4: Recommendation Systems

For your final task, you will use your knowledge of simple latent factor-based recommender systems to make predictions. Then evaluating the performance of your predictions.

Starting up

The next cell contains some starter code that you will need for your tasks in this section. Notice you are back to using the **trainingSet**.

In [46]:
```
#Create and fill our default dictionaries for our dataset
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)

for _, row in train_data.iterrows():
    user, item = row["customer_id"], row["product_id"]
    reviewsPerUser[user].append(row)
    reviewsPerItem[item].append(row)

#Create two dictionaries that will be filled with our rating prediction val
ues
userBiases = defaultdict(float)
itemBiases = defaultdict(float)

#Getting the respective lengths of our dataset and dictionaries
N = len(train_data)
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)

#Getting the list of keys
users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())

### You will need to use this list
y_rec = train_data['star_rating'].values
```

TODO 1: Calculate the ratingMean

1. Find the **average rating** of your training set.
2. Calculate a **baseline MSE value** from the actual ratings to the average ratings.

In [48]:
```
# Step 1: Calculate the average rating of the training set
ratingMean = train_data['star_rating'].mean()
print(f"Average Rating (ratingMean): {ratingMean:.2f}")

# Step 2: Calculate the baseline MSE (Baseline Mean Squared Error)
baseline_predictions = [ratingMean] * len(y_rec)

# Step 3: Calculate the baseline MSE
baseline_mse = mean_squared_error(y_rec, baseline_predictions)
print(f"Baseline MSE: {baseline_mse:.6f}")
```

```
Average Rating (ratingMean): 4.25
Baseline MSE: 1.480544
```

Here we are defining the functions you will need to optimize your MSE value.

In [50]:
```
#GIVEN
alpha = ratingMean

def prediction(user, item):
    return alpha + userBiases[user] + itemBiases[item]

def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    alpha = theta[0]
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    itemBiases = dict(zip(items, theta[1+nUsers:]))

def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(row["customer_id"], row["product_id"]) for _,
row in train_data.iterrows()]
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in userBiases:
        cost += lamb*userBiases[u]**2
    for i in itemBiases:
        cost += lamb*itemBiases[i]**2
    return cost

def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(train_data)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    for _, row in train_data.iterrows():
        user, item = row["customer_id"], row["product_id"]
        pred = prediction(user, item)
        diff = pred - row["star_rating"]
        dUserBiases[user] += 2 * diff / N
        dItemBiases[item] += 2 * diff / N
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
    for i in itemBiases:
        dItemBiases[i] += 2*lamb*itemBiases[i]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dItemBiases[i] f
or i in items]
    return numpy.array(dtheta)
```

TODO 2: Optimize

1. **Optimize** your MSE using the scipy.optimize.fmin_1_bfgs_b("arguments") functions.

In [52]:
```
# Initialize values
initial_alpha = ratingMean # Initial value for the global bias (alpha), set to the average rating
initial_userBiases = np.zeros(nUsers)  # Initialize user biases to 0
initial_itemBiases = np.zeros(nItems)  # Initialize item biases to 0
initial_theta = np.concatenate([[initial_alpha], initial_userBiases, initial_itemBiases])  # Combine all initial parameters into a single array (theta)

# Define the regularization parameter (lambda)
lamb = 0.1

# Optimize the MSE (Mean Squared Error) using the fmin_l_bfgs_b function
optimal_theta, optimal_cost, _ = fmin_l_bfgs_b(
    func=cost,  # Cost function to minimize
    x0=initial_theta,  # Initial values for optimization
    fprime=derivative,  # Derivative of the cost function (gradient)
    args=(y_rec, lamb),  # Additional arguments passed to the cost function (target values and regularization parameter)
    maxiter=100,  # Maximum number of iterations for the optimization
)

# Update model parameters with the optimized values
unpack(optimal_theta)

# Output results
print(f"Optimized alpha: {alpha}")
print(f"Final MSE: {optimal_cost:.6f}")
```

```
MSE = 1.4805442518035832
MSE = 1.4694341120672842
MSE = 1.4798262523165713
MSE = 1.4798261046628873
Optimized alpha: 4.2508145034156595
Final MSE: 1.480184
```

You're all done!

Congratulations! This project was the end of 4 whole courses worth of content! This project clearly didn't cover every single topic from those courses, but it serves as a summary for everything you have learned. This is only the start of Python Data Projects, so continue to learn and good luck in your future endeavors!