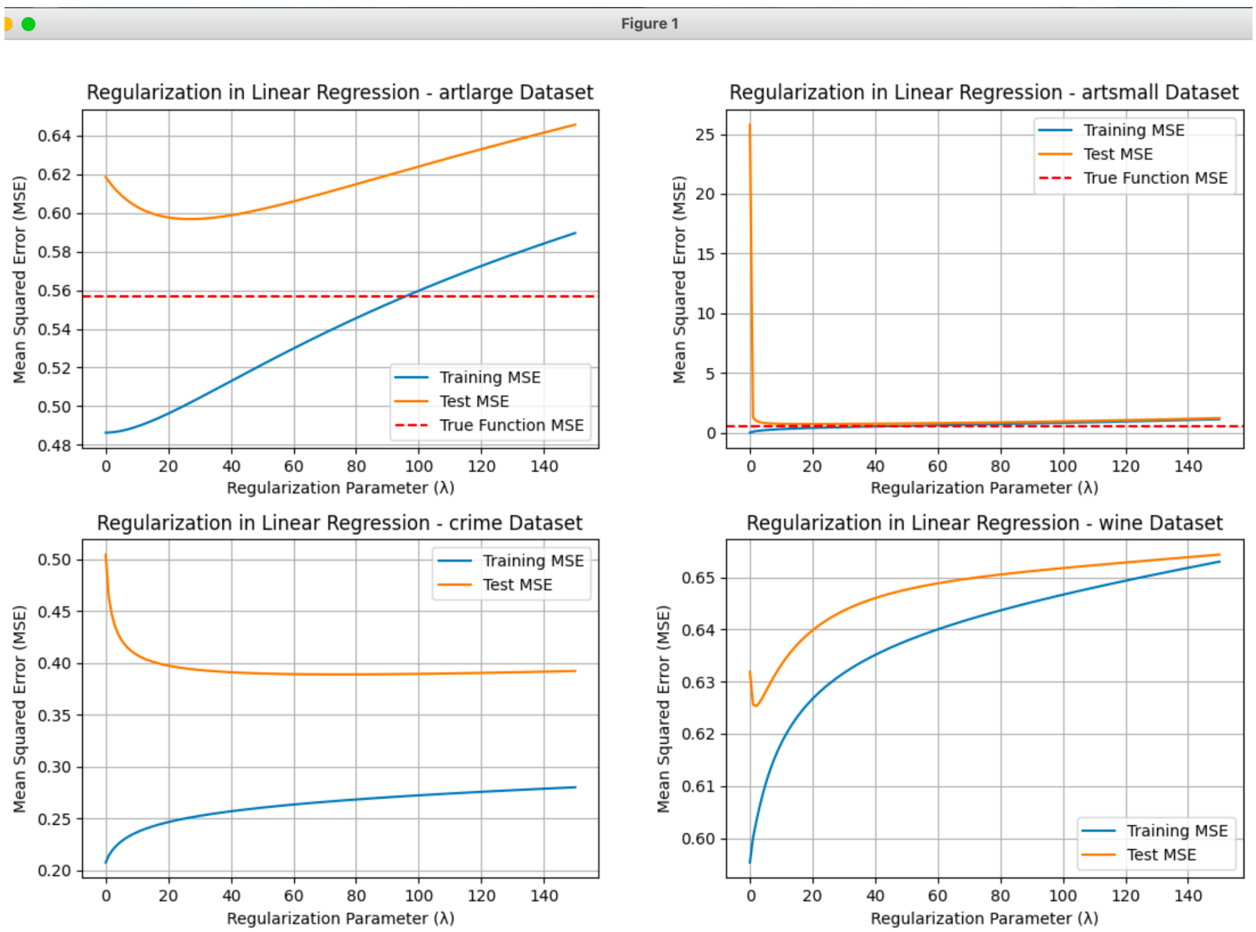
Sunilsakthivel Sakthi Velavan
Roni Khardon
CSCI-B 555
12 Oct.2023

<u>Programming Project 2</u>

Task 1:

For this experiment, 4 different plots were generated entailing the training and the testing MSE scores with respect to the λ values between 0 and 150. As seen below, the artificial datasets (artsmall and artlarge), we also compare the plotted MSEs with the MSE of the hidden true functions generating the data that give 0.533 (artsmall), and 0.557 (artlarge).

Plots:



Dataset	*λ of Lowest MSE*	*Lowest MSE*

artlarge	27	0.5967438457327094
artsmall	18	0.7202788056527154
crime	75	0.38902338771344425
wine	2	0.625308842304743

Observations:

For each dataset, one can make on the test MSE curves follow a pattern of a high initial MSE, followed by a quick drop to a minimum point followed by a steady growth as *λ* values from 0 to 150. Simultaneously, train MSE curves typically start off with a low MSE value and experience a quick increase to a point where the test MSE also experiences its lowest MSE and also followed by a slower increase (much like the tail end of the U-shaped test MSE curve). This actually explains why the training set MSE cannot be trusted to select the right λ values as when you use a small or zero value of λ, the model can fit the training data extremely well, resulting in a very low training set MSE. However, this does not necessarily mean that the model will generalize well to unseen data. The model might likely overfit the training data, capturing noise in the data, which will lead to poor performance on the test data. However, on the test set error, one can observe the U-shaped curve for all 4 datasets where a minimum MSE inflection point with an optimal λ value is reached after which larger λ values only increase the MSE. This suggests that the test MSE will be relatively high for small values of λ, indicating a model that overfits the training data. As λ increases, the test MSE decreases, suggesting better generalization and reduced overfitting. However, beyond a certain point, as λ becomes too large, the test MSE starts to increase, indicating underfitting or a too simple model.

While this fact can be stated in principle, the nuances of λ's impact on MSE actually vary between the 4 datasets. For instance, the minimal increase in λ for the artsmall dataset showcases how quickly the test MSE drops and converges at its true optimal value at an MSE of 0.533. This is in direct contrast to the artlarge dataset whose significantly larger size makes it more difficult to reach its true optimal λ value and its concurrent minimum MSE.

Task 2:

For this experiment, 10 fold cross validation was implemented on the training set to choose a λ value in the same range. We then retrained it on the entire training set and evaluated its performance on the training set. For parameter selection with a fixed train/test split, the data is first split into 10 disjoint portions, then each value of λ in the range V1, ..., VK, the following steps are performed: For each fold (i in 1...10), we train the ridge regression model on all portions except i. We then set the model on portion i and record the validation error for that fold. The average performance of λ on the 10 folds is recorded, the λ with the best average performance is selected, the model is then retrained on the entire training set (D) using the chosen λ, and the performance of the model with the selected λ is evaluated on the test set.

Results:

Dataset	*Selected λ*	*Validation MSE*	*Test MSE*	*Runtime (seconds)*
artlarge	27	0.5991366438451824	0.5967438457327094	3.0185658931732178
artsmall	16	0.703037732817224	0.7210310814686591	1.919619083404541
crime	150	0.344590450459746	0.39233899203438105	2.3310739994049072
wine	1	0.639636967500184	0.6256423273038403	0.5152280330657959

Observations:

For the "artlarge" dataset, Task 2 selected the same λ as in Task 1, and the test MSE remained very close. For the "artsmall" dataset, Task 2 chose a different λ compared to Task 1, and the test MSE was slightly higher in Task 2. For the "wine" dataset, Task 2 selected a different λ, and the test MSE was similar to Task 1. For the "crime" dataset, Task 2 selected a different λ, and the test MSE increased slightly compared to Task 1. However, the crime dataset is less visually actualized as the λ chosen here is 150 which suggests that the minimum point on the curve could be outside the range of [1, 150]. This is a relative possibility as the test MSE curve for crime could be argued to never quite reach its minimum within the same range either. However, overall, Task 2's approach allows for more flexibility in selecting λ based on cross-validation results.

Task 3:

Given the formulation of Bayesian linear regression with the simple prior w ~ N(0,(1/α)I), we can recall that the evidence function (and evidence approximation) gives a method to pick the parameters α and β. For this experiment, we implemented Bayesian Model Selection within the context of Baysian linear regression. This was done through integrating an iterative algorithm that converges and selects the ideal values of α and β using the training set to arrive at the values for m_n and SN, where mN is utilized to calculate the MSE on the test set for prediction. We initialized both α and β to random values in the range [1, 10] and a convergence threshold of 0.0001 that helps the algorithm ascertain that we've converged close enough to the actual values of α and β, and indirectly mN .

We first calculate the lambda values (λ_i) through determining the eigenvalues of the matrix ($\beta\Phi^T\Phi$). These eigenvalues are used in the calculations to update alpha and beta. Then, S_N is computed using equation (3.53), and m_N is calculated based on equation (3.54) which uses the value of S_N. These quantities represent the posterior covariance matrix and the posterior mean of the model coefficients, respectively. Next, using equations (3.91) and (3.92), we generate the values for gamma (using the old α and λ_i) and the new updated α. Then using equation (3.95), we generate the updated β (once again using the gamma dependent on the old α). Once the new α and β values are created, this algorithm reiterates until the difference between the old and new α and β values are less than the preset threshold value of .0001. Once the ideal α and β values have been converged upon, we can visualize the effective λ_i value can be computed as α/β and the m_N that was dependent on ideal β can be used to generate the predicted target value with which to generate the test MSE. Fundamentally, the task aims to find the optimal values of alpha and beta that result in the best predictions and minimal test MSE for Bayesian linear regression. It provides a Bayesian framework for model selection and parameter estimation.

Results:

Dataset	*α*	*β*	*λ (α/β)*	*Test MSE*	*Runtime*
artlarge	10.285784709807992	1.8603095111059353	5.52907172080908	0.6083085987853174	0.03248906135559082
artsmall	5.154543569445669	3.154402098963837	1.634079425428623	1.0635277958316793	0.0455479621887207
crime	425.6452824267614	3.250432133920129	130.95036748649758	0.39110230579177896	0.09948515892028809
wine	6.163857650413885	1.6098094296700247	3.828936231089999	0.6267461570340823	0.005588054656982422

Observations:

While for artlarge, artsmall and crime datasets, the λ values in Task 1 and Task 3 are drastically different with only wine having a close λ *value to* Task 1's. However, artlarge and artsmall both generated a significantly higher MSE compared to Task 1, indicating that the inclusion of α and β aren't particularly helpful. However, Task 3 achieved a lower test MSE compared to Task 1 on the crime dataset, suggesting improvement in predictive performance.

Discussion of Results:

In Task 2, the effective λ was determined as a result of cross-validation, which led to different λ values for each dataset. Also, Task 2 provided test MSE values that varied based on the λ selection through cross-validation. Task 3 calculated the effective λ as a result of the Bayesian model selection algorithm. This process also led to different λ values for each dataset and also provided test MSE values that varied based on the α and β selection through the Bayesian algorithm. The test set MSE values varied between Task 2 and Task 3. Task 3, the Bayesian model selection, tended to produce lower MSE values, indicating that it might lead to models with better predictive performance. However, this improvement in MSE might come at the cost of longer computation time. Comparatively, both Task 2 and Task 3 resulted in different effective λ values, suggesting that the choice of λ depends on the model selection method and dataset parameters. Overall, Task 2 seemed more suitable when computational efficiency is critical, and you want a data-driven approach for λ selection. It can work well when a quick model evaluation is needed, even if it might not yield the lowest possible test MSE. However, Task 3 is preferable when the focus is on achieving the best predictive performance. While it involves slightly longer run times, it leads to lower test MSE, which can be crucial in applications where accuracy is a lot more integral to the validity of the predicted values.

Codebase:

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import KFold
import time

datasets = ['artlarge', 'artsmall', 'crime', 'wine']

# Defines a range of regularization parameters (λ)
lambda_values = np.arange(0, 151, 1)

# Create subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
fig.tight_layout(pad=5.0)  # Add padding between subplots

def ridge_regression(X, y, lambda_val):
    N = X.shape[0]
    M = X.shape[1]

    # Calculate ΦᵀΦ
    phi_phi_transpose = np.dot(X.T, X)

    # Calculate λI
    lambda_I = lambda_val * np.identity(M)

    # Calculate (λI + ΦᵀΦ)⁻¹
    inv_term = np.linalg.inv(np.add(lambda_I, phi_phi_transpose))

    # Calculate Φᵀt
    phi_transpose_y = np.dot(X.T, y)

    # Calculate w using the equation: w = (λI + ΦᵀΦ)⁻¹Φᵀt
    w = np.dot(inv_term, phi_transpose_y)

    # Calculate predictions
    y_pred = np.dot(X, w)

    # Calculate Mean Squared Error (MSE)
    mse = np.sum((y - y_pred) ** 2) / N

    return w, mse

#Task 1: Regularization
print("Task 1:")
for i, dataset in enumerate(datasets):
    # Divide subplots into rows and columns
```

```python
    row = i // 2
    col = i % 2
    ax = axs[row, col]

    train_data = pd.read_csv('pp2data/train-'+dataset+'.csv', header=None)
    train_labels = pd.read_csv('pp2data/trainR-'+dataset+'.csv', header=None)
    test_data = pd.read_csv('pp2data/test-'+dataset+'.csv', header=None)
    test_labels = pd.read_csv('pp2data/testR-'+dataset+'.csv', header=None)


    X_train = train_data.values
    y_train = train_labels.values.ravel()
    X_test = test_data.values
    y_test = test_labels.values.ravel()


    # Lists to store MSE values
    train_mse = []
    test_mse = []

    # Iterate through lambda values and fit Ridge regression models
    for lambda_val in lambda_values:
        # Fit the Ridge regression model
        w, train_mse_val = ridge_regression(X_train, y_train, lambda_val)

        # Predict on training and test data
        y_test_pred = np.dot(X_test, w)

        # Calculate test set MSE
        test_mse_val = np.sum((y_test_pred - y_test) ** 2) / len(y_test)

        train_mse.append(train_mse_val)
        test_mse.append(test_mse_val)

    # Plot the results on the current subplot
    ax.plot(lambda_values, train_mse, label="Training MSE")
    ax.plot(lambda_values, test_mse, label="Test MSE")
    if dataset == 'artsmall':
        ax.axhline(y=0.533, color='r', linestyle='--', label="True Function MSE")
    elif dataset == 'artlarge':
        ax.axhline(y=0.557, color='r', linestyle='--', label="True Function MSE")
    ax.set_xlabel("Regularization Parameter (λ)")
    ax.set_ylabel("Mean Squared Error (MSE)")
    ax.set_title("Regularization in Linear Regression - "+dataset+" Dataset")
    ax.legend()
    ax.grid(True)
```

```
    print(f"Dataset: {dataset}")
    minMSElambda = test_mse.index(min(test_mse))
    print("Lambda:"+str(minMSElambda))
    print("Min MSE:"+str(min(test_mse)))
    print("\n")

# Show the entire figure with subplots
plt.show()


#Task 2

# Initialize variables to store results
best_lambda = {}  # Dictionary to store the best λ for each dataset
best_avg_validation_mse = {}  # Dictionary to store the best average
validation MSE for each dataset
test_mse = {}  # Dictionary to store the test MSE for each dataset
runtime = {}

for i, dataset in enumerate(datasets):
    train_data = pd.read_csv('pp2data/train-'+dataset+'.csv', header=None)
    train_labels = pd.read_csv('pp2data/trainR-'+dataset+'.csv', header=None)
    test_data = pd.read_csv('pp2data/test-'+dataset+'.csv', header=None)
    test_labels = pd.read_csv('pp2data/testR-'+dataset+'.csv', header=None)

    X_train = train_data.values
    y_train = train_labels.values.ravel()
    X_test = test_data.values
    y_test = test_labels.values.ravel()

    # Perform 10-fold cross-validation for λ selection
    kf = KFold(n_splits=10, shuffle=True, random_state=42)

    best_lambda[dataset] = None
    best_avg_validation_mse[dataset] = float('inf')
    test_mse[dataset] = None
    start_time = time.time()

    for lambda_val in lambda_values:
        validation_mses = []

        for train_index, val_index in kf.split(X_train):
            X_fold_train, X_fold_val = X_train[train_index], X_train[val_index]
            y_fold_train, y_fold_val = y_train[train_index], y_train[val_index]

            w, train_mse_val = ridge_regression(X_fold_train, y_fold_train,
lambda_val)
```

```
            # Predict on training and test data
            y_val_pred = np.dot(X_fold_val, w)

            # Calculate test set MSE
            validation_mse = np.sum((y_fold_val - y_val_pred) ** 2) /
len(y_fold_val)
            validation_mses.append(validation_mse)

        avg_validation_mse = np.mean(validation_mses)

        # Check if this λ has the lowest average validation MSE
        if avg_validation_mse < best_avg_validation_mse[dataset]:
            best_avg_validation_mse[dataset] = avg_validation_mse
            best_lambda[dataset] = lambda_val

            # Retrain the model on the entire training set with the best lambda
            w, test_mse_val = ridge_regression(X_train, y_train,
best_lambda[dataset])

            # Evaluate the model on the test set
            y_test_pred = np.dot(X_test, w)

            # Calculate test set MSE
            validation_mse = np.sum((y_test - y_test_pred) ** 2) / len(y_test)
            test_mse[dataset] = validation_mse

    end_time = time.time()
    runtime[dataset] = end_time - start_time

print("Task 2")
# Report results for all datasets
for dataset in datasets:
    print(f"Dataset: {dataset}")
    print(f"Selected λ: {best_lambda[dataset]}")
    print(f"Validation MSE with selected λ:
{best_avg_validation_mse[dataset]}")
    print(f"Test MSE with selected λ: {test_mse[dataset]}")
    print(f"Runtime: {runtime[dataset]} seconds")
    print("\n")

print("\nTask 3")
# Task 3: Bayesian Model Selection
for i, dataset in enumerate(datasets):
    train_data = pd.read_csv('pp2data/train-'+dataset+'.csv', header=None)
    train_labels = pd.read_csv('pp2data/trainR-'+dataset+'.csv', header=None)
    test_data = pd.read_csv('pp2data/test-'+dataset+'.csv', header=None)
    test_labels = pd.read_csv('pp2data/testR-'+dataset+'.csv', header=None)

    X_train = train_data.values
```

```
    y_train = train_labels.values.ravel()
    X_test = test_data.values
    y_test = test_labels.values.ravel()

    # Initialize alpha (α) and beta (β) to random values in the range [1, 10]
    alpha = np.random.uniform(1, 10)
    beta = np.random.uniform(1, 10)

    # Define convergence threshold
    threshold = 0.0001

    # Initialize variables to store results
    effective_lambda = None
    mse = None

    start_time = time.time()

    N = X_train.shape[0]
    M = X_train.shape[1]

    # Iterative algorithm to select α and β
    while True:
        lambda_values = np.linalg.eigvals(beta * np.dot(X_train.T, X_train))
        # Calculate SN and mN using equations (3.53) and (3.54)
        SN = np.linalg.inv((alpha * np.identity(M)) + (beta * np.dot(X_train.T,
X_train)))
        mN = beta * np.dot(np.dot(SN, X_train.T), y_train)

        # Update alpha and beta using equations (3.91) and (3.92)

        gamma = np.sum(lambda_values / (alpha + lambda_values))

        alpha_new = gamma / np.dot(mN.T, mN)

        y_pred = np.dot(X_train, mN)
        beta_new = (N - gamma)/(np.sum((y_train - y_pred) ** 2))

        # Check for convergence
        if abs(alpha_new - alpha) < threshold and abs(beta_new - beta) <
threshold:
            break

        alpha = alpha_new
        beta = beta_new

    # Calculate effective λ as α/β
    effective_lambda = alpha / beta

    mN = beta * np.dot(np.dot(SN, X_train.T), y_train)
```

```python
    # Use mN for prediction on the test set
    y_test_pred = np.dot(X_test, mN)

    # Calculate test set MSE
    mse = np.sum((y_test - y_test_pred) ** 2) / len(y_test)

    end_time = time.time()

    # Report results
    print(f"Dataset: {dataset}")
    print(f"Selected α: {alpha}")
    print(f"Selected β: {beta}")
    print(f"Effective λ (α/β): {effective_lambda}")
    print(f"Test MSE with selected α and β: {mse}")
    print(f"Runtime: {end_time - start_time} seconds")
    print('\n')
```

README

```
Just run the PP2.py file to get the 4 plots for the 4 dataset as per Task 1.
For further drilldown,
I also generated a table in the terminal with the lambda that generates the
lowest MSE for each dataset

For Task 2 and 3 to run, just close the window with the plots for the code to
go to the Task 2/3 section.
These sections will also print out the necessary information in the terminal
as per the assignment's instructions.
```