MDS5210 - Machine Learning Homework 3

Zijin CAI

December 12, 2024

Name: *Zijin CAI*
Student ID: *224040002*

Problem 1: Overfitting, Validation and Regularization

(a1)

given the training data $\{(x_i, y_i)\}_{i=1}^n$, denote $\Theta = (\theta_0, \ldots, \theta_8) \in \mathbb{R}^9$, and the linear model $\mathbf{y} = \mathbf{X}\Theta$

- $\mathbf{y}$ is a vector of the dependent variable value from the training data
- $\mathbf{X}$ is a matrix where each row corresponds to a data point, and each column corresponds to the term in polynomial.

```
data = loadmat('training_data.mat')
x_train = data['x'].flatten()
y_train = data['y'].flatten()
```

(a2)

the formulated least square problem: $\hat{\Theta} = \arg\min_{\Theta \in \mathbb{R}^9} \|\mathbf{X}\Theta - \mathbf{y}\|_2^2$
the plot of the fitted curve (limit x-axis from -1.5 to 1.5 and y-axis from -0.5 to 2.5) is shown in Figure 1

```
X_train = np.vander(x_train, 9, increasing=True)
theta_hat = np.linalg.lstsq(X_train, y_train, rcond=None)[0]

x_plot = np.linspace(x_train.min(), x_train.max(), 400)
X_plot = np.vander(x_plot, 9, increasing=True)
y_plot = X_plot.dot(theta_hat)

plt.figure(figsize=(8, 6))
plt.plot(x_plot, y_plot, 'r-', label='Fitted Curve', linewidth=2)
plt.scatter(x_train, y_train, color='blue', label='Training Data', zorder=5)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fitted Curve with 8th Order Polynomial')
plt.legend()
plt.xlim(-1.5, 1.5)
plt.ylim(-0.5, 2.5)
plt.grid(True)
plt.show()
```

```
The estimated theta_hat:
array([  0.60775979,  -7.25862115,  15.3450059 ,  17.26526602,  -46.39728808,
    -10.63103168,  41.47282583,   1.75929095,  -11.22913232])
```

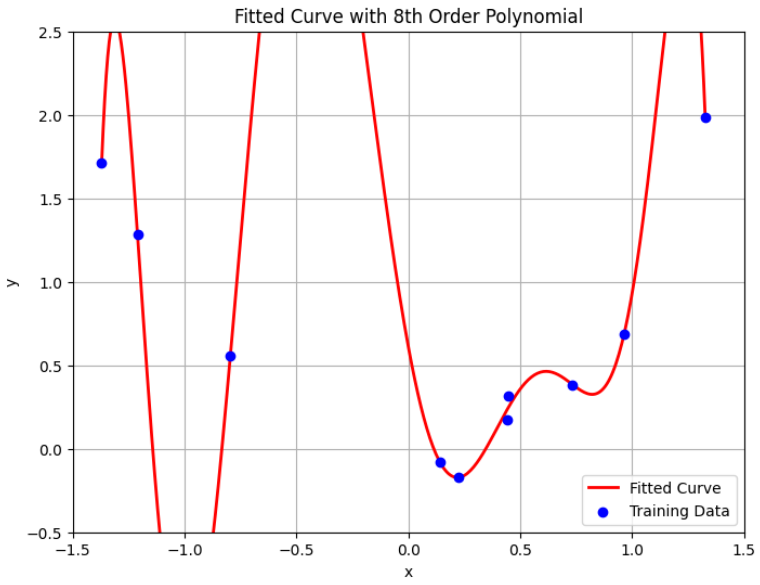

Figure 1: Fitted curve with polynomial degree 8

(a3)

given the test dataset, the test error $\left\|\mathbf{X}_{test}\hat{\Theta} - \mathbf{y}_{test}\right\|_2$ can be thus calculated

```
test_data = loadmat('test_data.mat')

x_test = test_data['x_test'].flatten()
y_test = test_data['y_test'].flatten()

X_test = np.vander(x_test, 9, increasing=True)
y_pred = X_test.dot(theta_hat)
test_error = np.linalg.norm(y_pred - y_test)
print(f"Test Error: {test_error}")
```

```
Test Error:  4.560738896701371
```

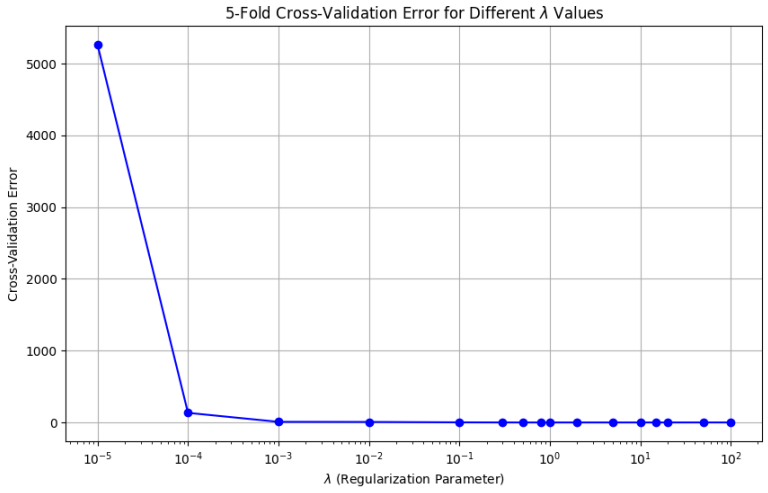

Figure 2: 5-Fold Cross-Validation Error against Different λ

(b1)

the underlying model is quadratic, while the fitted model is of degree 8, so it is overfitting.
to prevent overfitting, we can add a regularization term to the least square problem (l_2-regularized least square):

$$\hat{\Theta} = \arg\min_{\Theta \in \mathbb{R}^9} \|\mathbf{X}\Theta - \mathbf{y}\|_2^2 + \lambda \|\Theta\|_2^2 \tag{1}$$

the set of candidates λ is $\left\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 0.3, 0.5, 0.8, 1, 2, 5, 10, 15, 20, 50, 100\right\}$ for selecting regularization parameter λ using 5-fold cross-validation, and the plot of validation error against λ is shown in Figure 2

```
lambdas = [10**-5, 10**-4, 10**-3, 10**-2, 10**-1,
        0.3, 0.5, 0.8, 1, 2, 5, 10, 15, 20, 50, 100]

kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_errors = []

for lam in lambdas:
    ridge = Ridge(alpha=lam)
    fold_errors = []
    for train_index, test_index in kf.split(X_train):
        X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
        y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

        ridge.fit(X_train_fold, y_train_fold)
        y_pred_fold = ridge.predict(X_test_fold)
        fold_errors.append(np.mean((y_pred_fold - y_test_fold)**2))
    cv_errors.append(np.mean(fold_errors))

plt.figure(figsize=(10, 6))
plt.plot(lambdas, cv_errors, marker='o', linestyle='-', color='b')
plt.xscale('log')  # Set the x-axis to log-scale
plt.xlabel(r'$\lambda$ (Regularization Parameter)')
plt.ylabel('Cross-Validation Error')
plt.title('5-Fold Cross-Validation Error for Different $\lambda$ Values')
plt.grid(True)
plt.show()
```

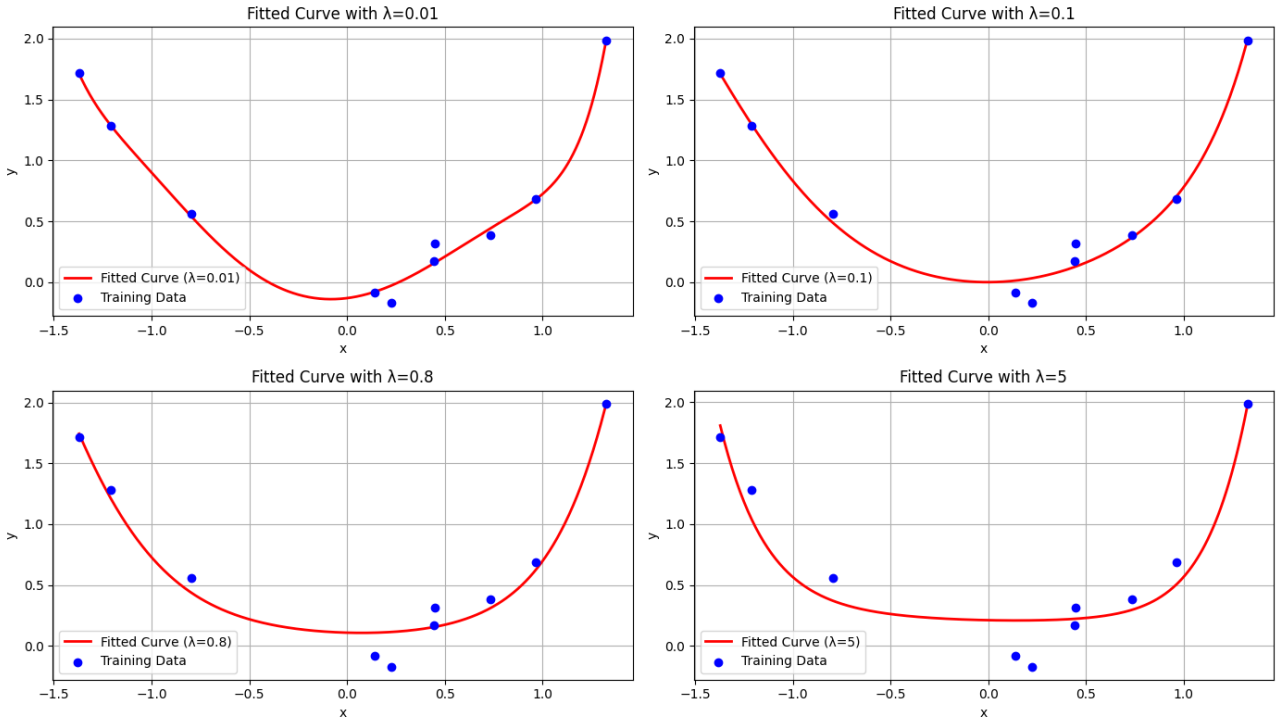

Figure 3: Fitted curve with different λ

(b2)

reset the regularization parameter λ to $0.01, 0.1, 0.8, 5$ and solve for the corresponding $\hat{\Theta}$, then plot the fitted curve for each λ in Figure 3

```
lambdas_to_plot = [0.01, 0.1, 0.8, 5]

x_plot = np.linspace(x_train.min(), x_train.max(), 400)
X_plot = np.vander(x_plot, 9, increasing=True)

plt.figure(figsize=(14, 8))

for lam in lambdas_to_plot:
    ridge = Ridge(alpha=lam)
    ridge.fit(X_train, y_train)
    y_pred = ridge.predict(X_plot)

    plt.subplot(2, 2, lambdas_to_plot.index(lam) + 1)
    plt.plot(x_plot, y_pred, 'r-', label=f'Fitted Curve (={lam})', linewidth=2)
    plt.scatter(x_train, y_train, color='blue', label='Training Data', zorder=5)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(f'Fitted Curve with ={lam}')
    plt.legend()
    plt.grid(True)

plt.tight_layout()
plt.show()
```

(b3)

the test error $\left\|\mathbf{X}_{test}\hat{\Theta} - \mathbf{y}_{test}\right\|_2$ for each λ is calculated as follows:

```
lambdas_to_evaluate = [0.01, 0.1, 0.8, 5]
test_errors = []

for lam in lambdas_to_evaluate:
    ridge = Ridge(alpha=lam)
    ridge.fit(X_train, y_train)  # Re-fit the model with the current lambda
    y_pred = ridge.predict(X_test)
    test_errors.append(np.mean((y_pred - y_test) ** 2))

print("Test Errors for each lambda:")
for lam, error in zip(lambdas_to_evaluate, test_errors):
    print(f"Lambda: {lam}, Test MSE: {error}")
```

```
Lambda: 0.01, Test MSE: 0.04410926529975627
Lambda: 0.1, Test MSE: 0.04021179004610607
Lambda: 0.8, Test MSE: 0.040551902406536354
Lambda: 5, Test MSE: 0.05019667263268211
```

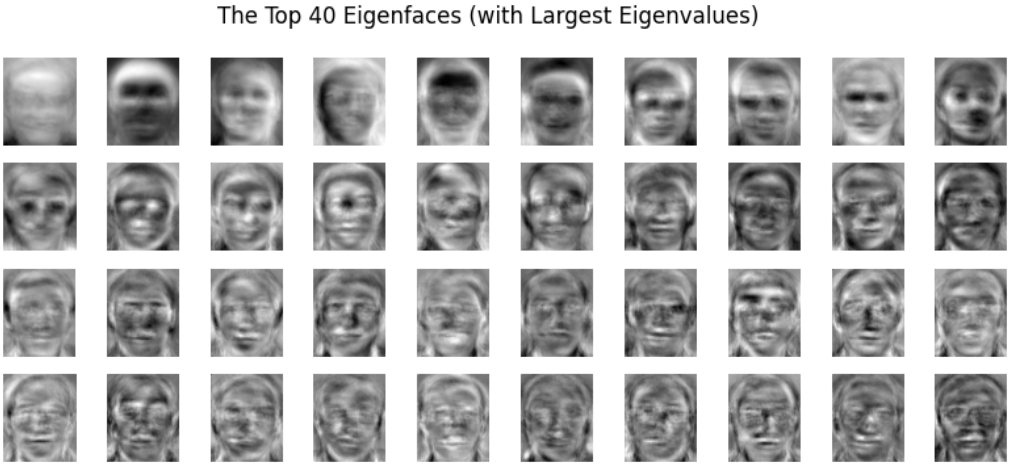

Figure 4: The Top 40 Eigenfaces (with Largest Eigenvalues)

Problem 2: Face Reconstruction by PCA

the principal component analysis (PCA) projects high-dimensional data into lower-dimensional by optimizing:

$$\min_{A^\top A=I,\Theta} \|\mathbf{X} - \mathbf{A}\mathbf{\Theta}\|_F^2. \tag{2}$$

the ORL dataset consists of 400 images of 40 people, each person with 10 images, of size 92×112 pixels

- $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ is the data matrix, where $d = 10304$ and $n = 400$
- $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_k] \in \mathbb{R}^{d\times k}$ is the basis matrix, which can be interpreted as the prototype of faces (eigenfaces)
- $\mathbf{\Theta} = [\theta_1, \ldots, \theta_n] \in \mathbb{R}^{k\times n}$ is the principal components, and the choice of k determines the quality of the reconstructed images
- the i-th reconstructed image $\mathbf{A}\mathbf{\Theta}_i$ is the linear combination of these eigenfaces: $\sum_{j=1}^{k} \mathbf{a}_j \theta_i[j]$

(1)

the eigenfaces with the top 40 largest eigenvalues are shown in Figure 4.

```
from p2_release import *
import scipy.linalg as la

X_mean = np.mean(X_processed, axis=1)
X_centered = X_processed - X_mean
U, S, Vt = la.svd(X_centered, full_matrices=False)

plt.figure(figsize=(10, 4))
for i in range(40):
    plt.subplot(4, 10, i + 1)
    plt.imshow(U[:, i].reshape(img_size), cmap='gray')
    plt.axis('off')
plt.suptitle("The Top 40 Eigenfaces (with Largest Eigenvalues)")
plt.show()
```

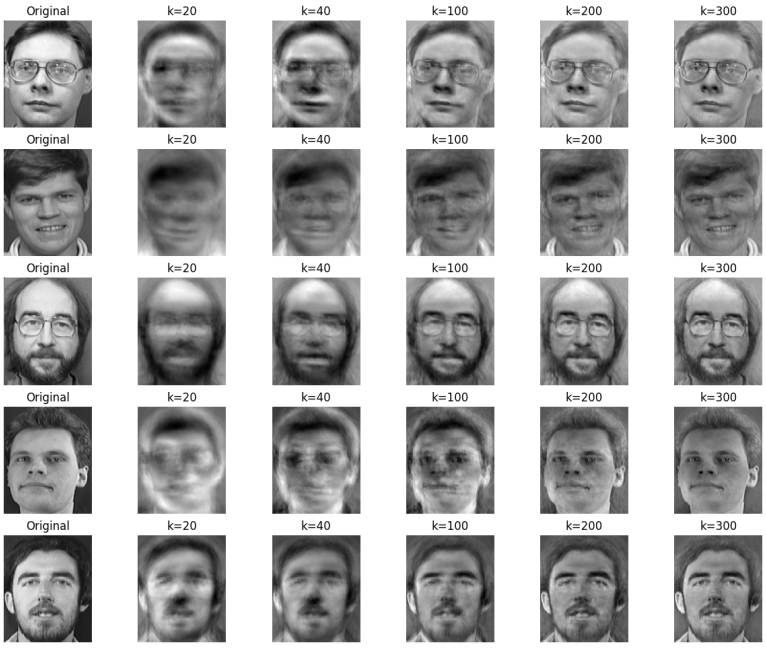

Figure 5: The Reconstructed Images with Different k

(2)

the corresponding reconstructed images with $k \in \{20, 40, 100, 200, 300\}$ and the original are shown in Figure 5

```
indices = np.random.choice(400, 5, replace=False)
original_images = X_processed[:, indices]
k_values = [20, 40, 100, 200, 300]

fig, axes = plt.subplots(5, 6, figsize=(15, 12))
for i, idx in enumerate(indices):
    axes[i, 0].imshow(X_processed[:, idx].reshape(img_size), cmap='gray')
    axes[i, 0].set_title('Original')
    axes[i, 0].axis('off')

    for j, k in enumerate(k_values):
        X_recon = np.dot(U[:, :k], np.dot(np.diag(S[:k]), Vt[:k, :]))
        axes[i, j+1].imshow(X_recon[:, idx].reshape(img_size), cmap='gray')
        axes[i, j+1].axis('off')
plt.show()
```

Apparently, increasing k generally improves the quality of the reconstructed images by capturing more of the data's variance, which can be responsible for that the PCA is a lossy compression method, and more information is retained with a larger k. However, there is a point of diminishing returns, and practical considerations such as computational efficiency and the need for a compact representation may dictate the optimal choice of k

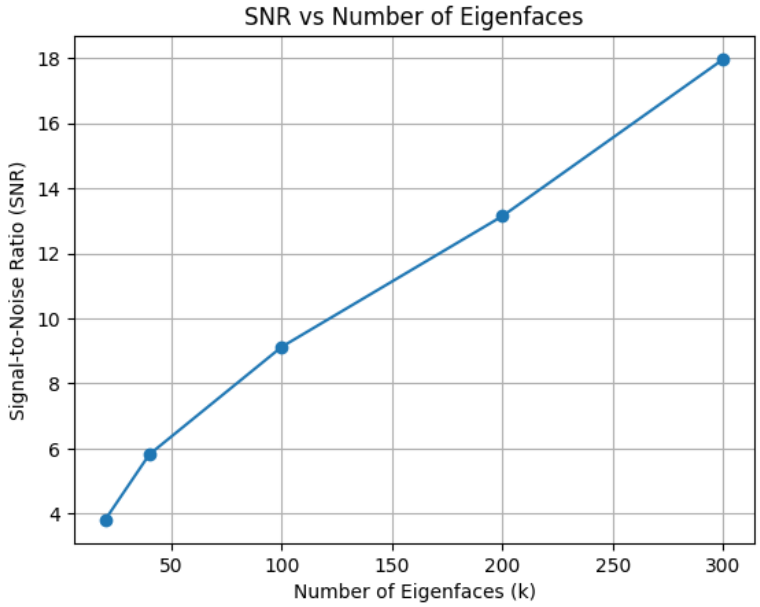

Figure 6: The SNR Ratio under Different Choices of k

(3)

the signal-to-noise ratio (SNR) is defined as:

$$\text{SNR} = 10 \cdot \log_{10} \left(\frac{\|\mathbf{X}_{\text{recon}}\|_F^2}{\|\mathbf{X}_{\text{recon}} - \mathbf{X}\|_F^2} \right) \tag{3}$$

where $\mathbf{X}_{\text{recon}}$ is the reconstructed image matrix and $\mathbf{X}$ is the original image matrix and the plot of SNR under different choices of k is shown in figure 6

```
snr_values = []
for k in k_values:
    X_recon = np.dot(U[:, :k], np.dot(np.diag(S[:k]), Vt[:k, :]))
    snr = cal_snr(X_centered, X_recon)
    snr_values.append(snr)

plt.plot(k_values, snr_values, marker='o')
plt.xlabel('Number of Eigenfaces (k)')
plt.ylabel('Signal-to-Noise Ratio (SNR)')
plt.title('SNR vs Number of Eigenfaces')
plt.show()
```

Problem 3: Support Vector Machine and Kernel Methods

The MNIST dataset contains approximately 7000 samples for each class, split with 4000 samples for training set and remain samples for testing set. The training set is split into 3000 for training and 1000 for validation, and to accelerate the experiment, select 300 for training and 100 for validation (10% of the data)

(a)

train an SVM with the inhomogeneous linear and quadratic kernels such that:

$$k(\mathbf{u}, \mathbf{v}) = \left(\mathbf{u}^\top \mathbf{v} + 1\right)^p, \quad p = 1, 2 \tag{4}$$

```
    X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
    X, y = X / 255.0, y.astype(int)
    X_train, y_train, X_val, y_val, X_test, y_test = sample_class(X, y, 300, 100, 300)
    C_values = np.logspace(-3, 3, 10)

    linear_svm = SVC(kernel='poly', degree=1)
    linear_grid = GridSearchCV(linear_svm, {'C': C_values}, cv=5)
    linear_grid.fit(X_train, y_train)
    best_C_linear = linear_grid.best_params_['C']
    val_error_linear = 1 - linear_grid.best_score_

    quadratic_svm = SVC(kernel='poly', degree=2)
    quadratic_grid = GridSearchCV(quadratic_svm, {'C': C_values}, cv=5)
    quadratic_grid.fit(X_train, y_train)
    best_C_quadratic = quadratic_grid.best_params_['C']
    val_error_quadratic = 1 - quadratic_grid.best_score_

    plt.figure(figsize=(10, 5))
    plt.plot(C_values, 1 - np.array(linear_grid.cv_results_['mean_test_score']),
            label='Linear Kernel (degree=1)', marker='o', color='blue')
    plt.plot(C_values, 1 - np.array(quadratic_grid.cv_results_['mean_test_score']),
            label='Quadratic Kernel (degree=2)', marker='o', color='red')
    plt.xscale('log')
    plt.xlabel('C')
    plt.ylabel('Validation Error')
    plt.legend()
    plt.title('Validation Error vs C for Linear and Quadratic Kernels')
    plt.show()
```

```
final_linear_svm = SVC(kernel='poly', degree=1, C=best_C_linear)
final_linear_svm.fit(np.concatenate([X_train, X_val]), np.concatenate([y_train, y_val]))
test_error_linear = 1 - final_linear_svm.score(X_test, y_test)

final_quadratic_svm = SVC(kernel='poly', degree=2, C=best_C_quadratic)
final_quadratic_svm.fit(np.concatenate([X_train, X_val]),
                        np.concatenate([y_train, y_val]))
test_error_quadratic = 1 - final_quadratic_svm.score(X_test, y_test)

print(f"Linear Kernel (degree=1): Best C = {best_C_linear},
  Validation Error = {val_error_linear:.4f}, Test Error = {test_error_linear:.4f}")
print(f"Quadratic Kernel (degree=2): Best C = {best_C_quadratic},
  Validation Error = {val_error_quadratic:.4f}, Test Error = {test_error_quadratic:.4f}")
```

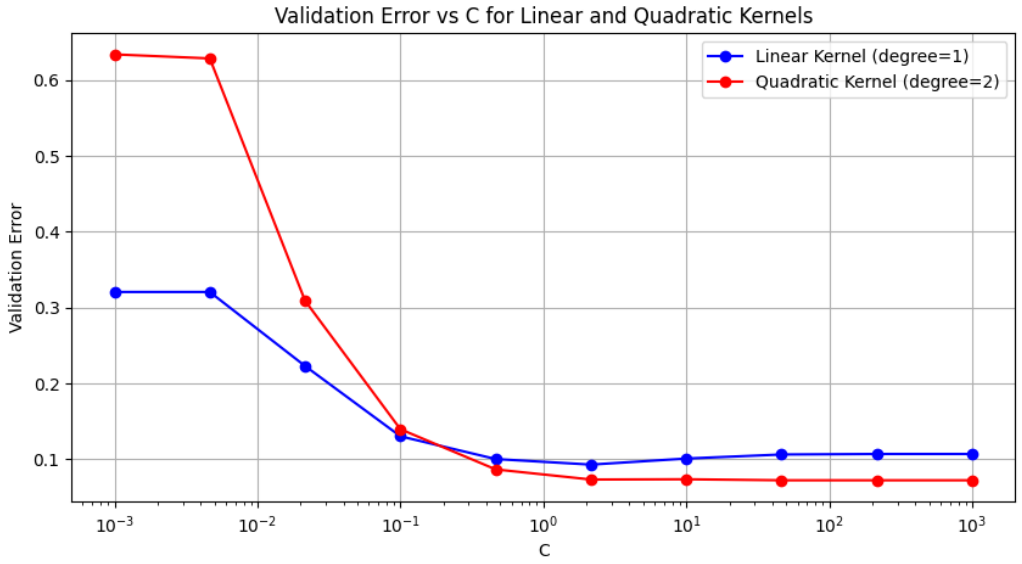

Figure 7: Validation Error and C for Linear and Quadratic Kernels

```
Linear Kernel (degree=1):
        Best C = 2.154434690031882, Validation Error = 0.0927, Test Error = 0.0806
Quadratic Kernel (degree=2):
        Best C = 46.41588833612773, Validation Error = 0.0720, Test Error = 0.0540
```

(b)

repeat the experiment with the radial basis function (RBF) kernel:

$$k(\mathbf{u}, \mathbf{v}) = \exp\left(\|\mathbf{u} - \mathbf{v}\|^2\right) \tag{5}$$

```
    gamma_values = [1e-4, 1e-3, 1e-2, 1e-1, 1e-0, 1e1]
    param_grid_rbf = {'C': C_values, 'gamma': gamma_values}

    rbf_svm = SVC(kernel='rbf')
    rbf_grid = GridSearchCV(rbf_svm, param_grid_rbf, cv=5)
    rbf_grid.fit(X_train, y_train)
    best_params_rbf = rbf_grid.best_params_
    best_C_rbf = best_params_rbf['C']
    best_gamma_rbf = best_params_rbf['gamma']
    val_error_rbf = 1 - rbf_grid.best_score_

    plt.figure(figsize=(10, 5))
    scores_matrix = rbf_grid.cv_results_['mean_test_score'].
        reshape(len(C_values), len(gamma_values))
    for i, gamma in enumerate(gamma_values):
        plt.plot(C_values, 1 - scores_matrix[:, i], label=f'Gamma={gamma}', marker='o')

    plt.xscale('log')
    plt.xlabel('C')
    plt.ylabel('Validation Error')
    plt.title('Validation Error vs C for RBF Kernel with specific gamma values')
    plt.legend()
    plt.show()
```

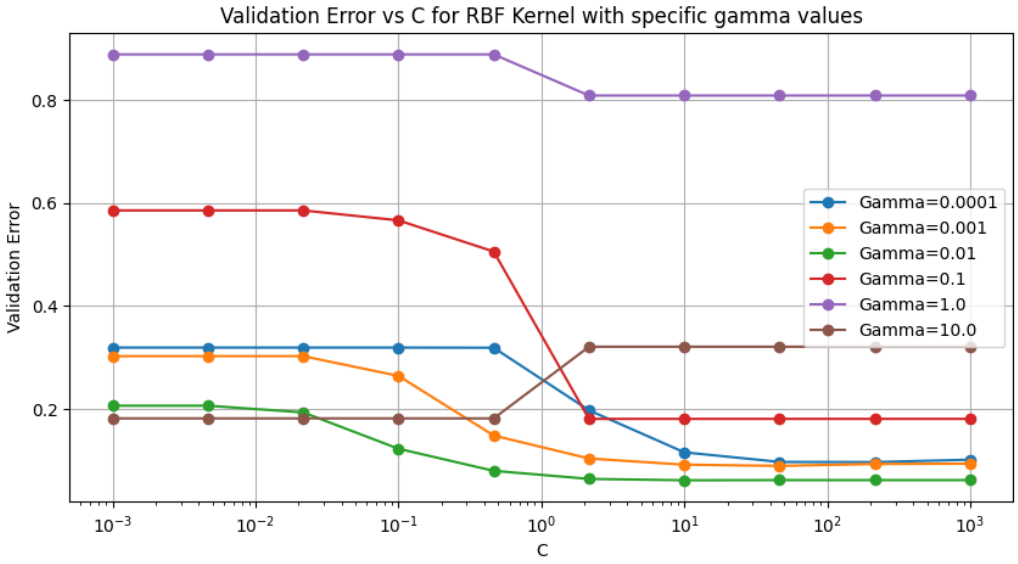

Figure 8: Validation Error and C for RBF Kernel with specific gamma values

```
final_rbf_svm = SVC(kernel='rbf', C=best_C_rbf, gamma=best_gamma_rbf)
final_rbf_svm.fit(np.concatenate([X_train, X_val]), np.concatenate([y_train, y_val]))
test_error_rbf = 1 - final_rbf_svm.score(X_test, y_test)

print(f"RBF Kernel: Best C = {best_C_rbf}, Best Gamma = {best_gamma_rbf},
        Validation Error = {val_error_rbf:.4f}, Test Error = {test_error_rbf:.4f}")
```

```
RBF Kernel: Best C = 10.0, Best Gamma = 0.01, Validation Error = 0.0610, Test Error = 0.0494
```