HW4

Octi Zhang

November 28, 2023

OCTIPUS

1

a

True. This is because the rank of the matrix (k = rank (X)) represents the maximum number of linearly independent vectors in the matrix. PCA finds the best linear subspace of a given dimensionality (in this case, k) that captures the most variance in the data. If the dimensionality of the subspace is equal to the rank of the data matrix, it means that the subspace can capture all the variations in the data, leading to no information loss in the projection.

b

False. $X = USV^T$ then $X^TX = VS^TU^TUSV^T = VS^TSV^T$, due to X^TX being symmetric, it guarantees it has eigenvalue and eigenvectors such that $X^TXv = \lambda v$, therefore $VS^TSV^Tv = \lambda v$, if we let $v = V$, then $VS^TS = \lambda V$. Since S^TS is diagonal, it shows that λV is equivalent as scaling each column of V with S^TS. Therefore it is the column of V that are equal to the eigenvectors of X^TX not rows

c

False. Minimizing the k-means objective function by increasing k can lead to overfitting, where clusters may become arbitrarily small and not necessarily meaningful. It is better to use methods like the elbow method or silhouette score to determine an appropriate k that balances the granularity of clustering with the overall cluster quality.

d

False, if there are repeated value s in S, then the row and column in U and V associated with s can have multiple valid order, this shows that USV decomposition are not unique.

e

False, the rank of a square matrix is determined by the number of linearly independent rows or columns. However, a linearly independent matrix may have repeated eigenvalues and results in number of unique eigenvalues less than its rank,

2

a

Data Pre-processing Steps:

- **Data Cleaning:** First, Handle missing values, duplicates, and outliers in the dataset.
- **Encoding Categorical Data:** Then, Convert categorical variables into numerical values using one-hot encoding or label encoding.
- **Normalization/Standardization:** Scale numerical data to ensure fair contribution to the model.
- **Missing Data Interpolation:** For missing entries, use methods like k-nearest neighbors, mean/mode imputation, or model-based imputation where appropriate.
- **Balancing Classes:** Once dataset is complete, check if the dataset is imbalanced, apply oversampling, undersampling, or bootstraping.

Machine Learning Pipeline Steps:

- **Train Test Validation Split:** split train, test, validation with 70%, 20%, 10%.
- **Feature Selection:** Employ L1 regularization (Lasso) to help in feature selection and to create a sparse model emphasizing important risk factors.
- **Model Selection:** Start with logistic regression for binary natured classification (disease/no disease). If non-linear relationships are suspected, consider kernel for quicker compute, and deep learning techniques.
- **Validation Technique:** Use k-fold cross-validation to assess the generalizability of the model.
- **Hyperparameter Tuning:** Use grid search or random search with cross-validation to find optimal hyperparameters.

Acknowledging Constraints and Measuring Results:

- **Accuracy over Efficiency:** Focus on a model that provides more accurate result as they are preciseness of infomation creates huge matter to the user.
- **Privacy Considerations:** Ensure that the model does not require or store more personal data than necessary, respecting user privacy.

b

Shortcomings of Training Process:

Limited representation in the training dataset can lead to skewed accuracy across different demographic groups. Features may inadvertently emphasize specific characteristics of the majority population in the dataset. Moreover, Over-representation of certain classes (e.g., more non-disease cases) might result in biased predictions.

Addressing the Shortcomings:

1. Include data from varied demographics to ensure representativeness. 2. plot the data base one the feature and remove data that are overly populated in the same cluster, make sure the training data is equally balanced before training 3. Apply regularization and stratified k-fold cross-validation for robust performance evaluation.

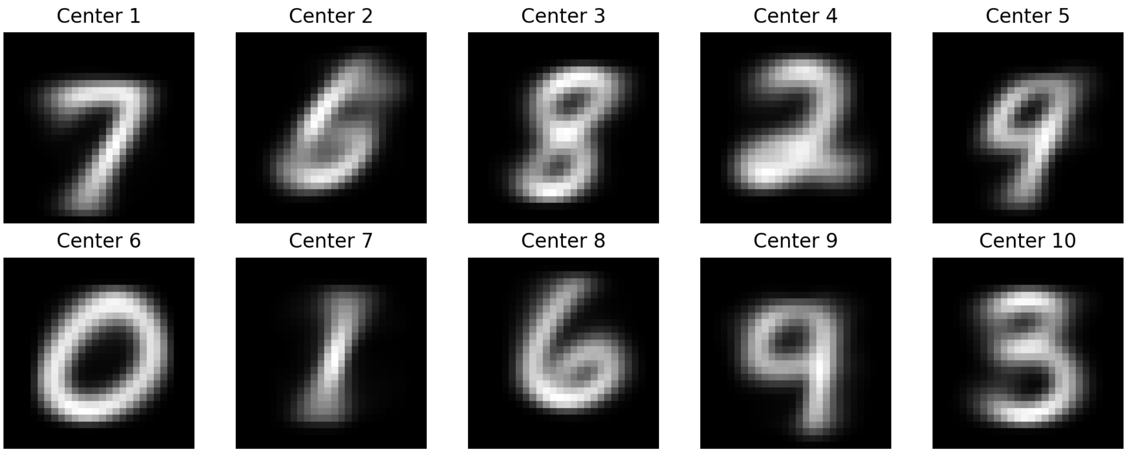
c

Ignoring shortcomings in crime datasets can lead to biased predictive policing, disproportionately affecting minority communities. This can misguide resource allocation, favoring increased law enforcement over necessary social services in areas needing them. Such practices erode public trust in law enforcement, potentially reducing crime reporting and exacerbating data inaccuracies. Relying on skewed data raises legal and ethical issues due to the unfair targeting of specific demographics and perpetuates stereotypes, negatively impacting the socio-economic fabric of communities.

3

a

b



```python
from typing import List, Tuple

import numpy as np
import itertools
from utils import problem


@problem.tag("hw4-A")
def calculate_centers(
    data: np.ndarray, classifications: np.ndarray, num_centers: int
) -> np.ndarray:
    """
    Sub-routine of Lloyd's algorithm that calculates the centers given datapoints and their respecti
    num_centers is additionally provided for speed-up purposes.

    Args:
        data (np.ndarray): Array of shape (n, d). Training data set.
        classifications (np.ndarray): Array of shape (n,) full of integers in range {0, 1, ...,  num_
            Data point at index i is assigned to classifications[i].
        num_centers (int): Number of centers for reference.
            Might be usefull for pre-allocating numpy array (Faster that appending to list).

    Returns:
        np.ndarray: Array of shape (num_centers, d) containing new centers.
    """
    d = data.shape[1]
    cluster_sums = np.zeros((num_centers, d))
    cluster_counts = np.zeros(num_centers)

    for i in range(len(data)):
        cluster_sums[classifications[i]] += data[i]
        cluster_counts[classifications[i]] += 1

    # Avoid division by zero for empty clusters
    cluster_counts[cluster_counts == 0] = 1

    new_centers = cluster_sums / cluster_counts[:, None]

    return new_centers



@problem.tag("hw4-A")
def cluster_data(data: np.ndarray, centers: np.ndarray) -> np.ndarray:
    """
    Sub-routine of Lloyd's algorithm that clusters datapoints to centers given datapoints and centers

    Args:
        data (np.ndarray): Array of shape (n, d). Training data set.
        centers (np.ndarray): Array of shape (k, d). Each row is a center to which a datapoint can be

    Returns:
        np.ndarray: Array of integers of shape (n,), with each entry being in range {0, 1, 2, ..., k
            Entry j at index i should mean that j^th center is the closest to data[i] datapoint.
    """

    distances = np.zeros((data.shape[0], centers.shape[0]))
    for idx, center in enumerate(centers):
        distances[:, idx] = np.sqrt(np.sum((data - center) ** 2, axis=1))
    return np.argmin(distances, axis=1)

def calculate_error(data: np.ndarray, centers: np.ndarray) -> float:
    """ This method has been implemented for you.

    Calculates error/objective function on a provided dataset, with trained centers.

    Args:
        data (np.ndarray): Array of shape (n, d). Dataset to evaluate centers on.
        centers (np.ndarray): Array of shape (k, d). Each row is a center to which a datapoint can be
            These should be trained on training dataset.

    Returns:
        float: Single value representing mean objective function of centers on a provided dataset.
    """
    distances = np.zeros((data.shape[0], centers.shape[0]))
    for idx, center in enumerate(centers):
        distances[:, idx] = np.sqrt(np.sum((data - center) ** 2, axis=1))
    return np.mean(np.min(distances, axis=1))


@problem.tag("hw4-A")
def lloyd_algorithm(
    data: np.ndarray, num_centers: int, epsilon: float = 10e-3
) -> Tuple[np.ndarray, List[float]]:
    """Main part of Lloyd's Algorithm.

    Args:
        data (np.ndarray): Array of shape (n, d). Training data set.
        num_centers (int): Number of centers to train/cluster around.
        epsilon (float, optional): Epsilon for stopping condition.
```

```
            Training should stop when max(abs(centers - previous_centers)) is smaller or equal to eps
            Defaults to 10e-3.

    Returns:
        np.ndarray: Tuple of 2 numpy arrays:
            Element at index 0: Array of shape (num_centers, d) containing trained centers.
            Element at index 1: List of floats of length # of iterations
                containing errors at the end of each iteration of lloyd's algorithm.
                You should use the calculate_error() function that has been implemented for you.

    Note:
        - For initializing centers please use the first `num_centers` data points.
    """
    last_trained_center = np.ones((num_centers, data.shape[1]))
    trained_center = data[np.random.choice(np.arange(data.shape[0]), num_centers, False)]
    errors = []
    while (np.max(abs(trained_center - last_trained_center)) > epsilon):
        clustered_data = cluster_data(data, trained_center)
        last_trained_center = trained_center
        trained_center = calculate_centers(data=data, classifications=clustered_data, num_centers=num
        errors.append(calculate_error(data=data, centers=trained_center))
    return (trained_center, errors)




if __name__ == "__main__":
    from k_means import lloyd_algorithm  # type: ignore
else:
    from .k_means import lloyd_algorithm

import matplotlib.pyplot as plt
import numpy as np

from utils import load_dataset, problem


@problem.tag("hw4-A", start_line=1)
def main():
    """Main function of k-means problem

    Run Lloyd's Algorithm for k=10, and report 10 centers returned.

    NOTE: This code might take a while to run. For debugging purposes you might want to change:
        x_train to x_train[:10000]. CHANGE IT BACK before submission.
    """
    (x_train, _), _ = load_dataset("mnist")
    centers = lloyd_algorithm(x_train, num_centers=10, epsilon=10e-3)
    # Reshape centers to 28x28 (assuming MNIST images are 28x28)
    centers_images = centers[0].reshape(-1, 28, 28)

    # Plotting the centers
    fig, axes = plt.subplots(2, 5, figsize=(10, 4))
    for i, ax in enumerate(axes.flatten()):
        ax.imshow(centers_images[i], cmap='gray')
        ax.axis('off')
        ax.set_title(f'Center {i+1}')
    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()
```

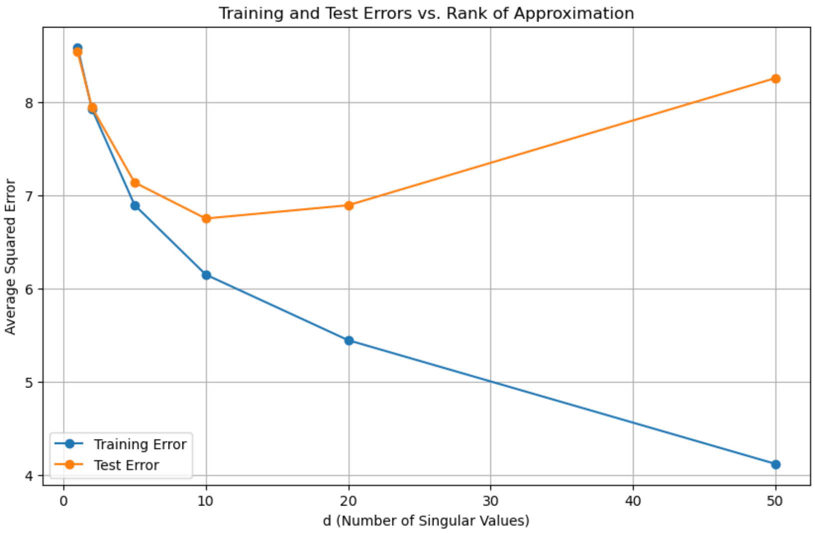
4

a

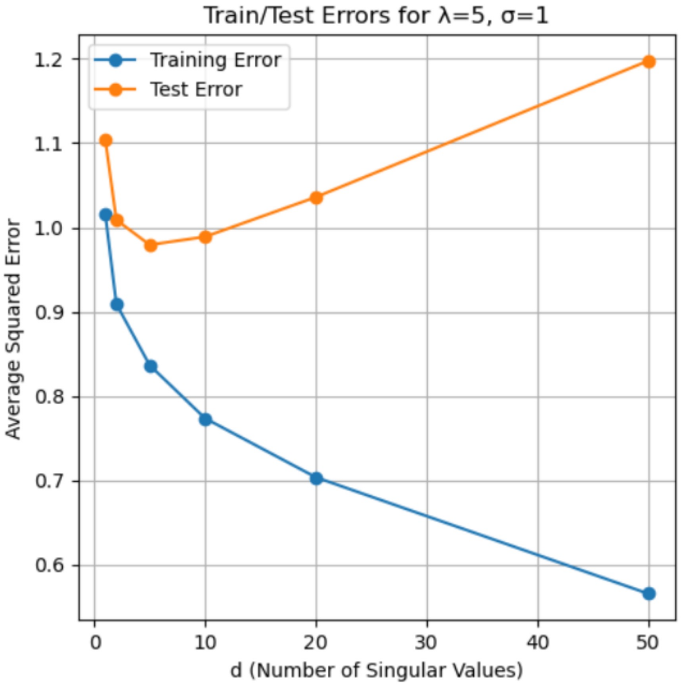
$$\hat{R} = \frac{\sum_{alli}(R_{ij}|R_{ij_1} = R_{ij_2})}{num(R_{ij}|R_{ij_1} = R_{ij_2})} forj = 0, 1, 2, \ldots, movies - 1$$

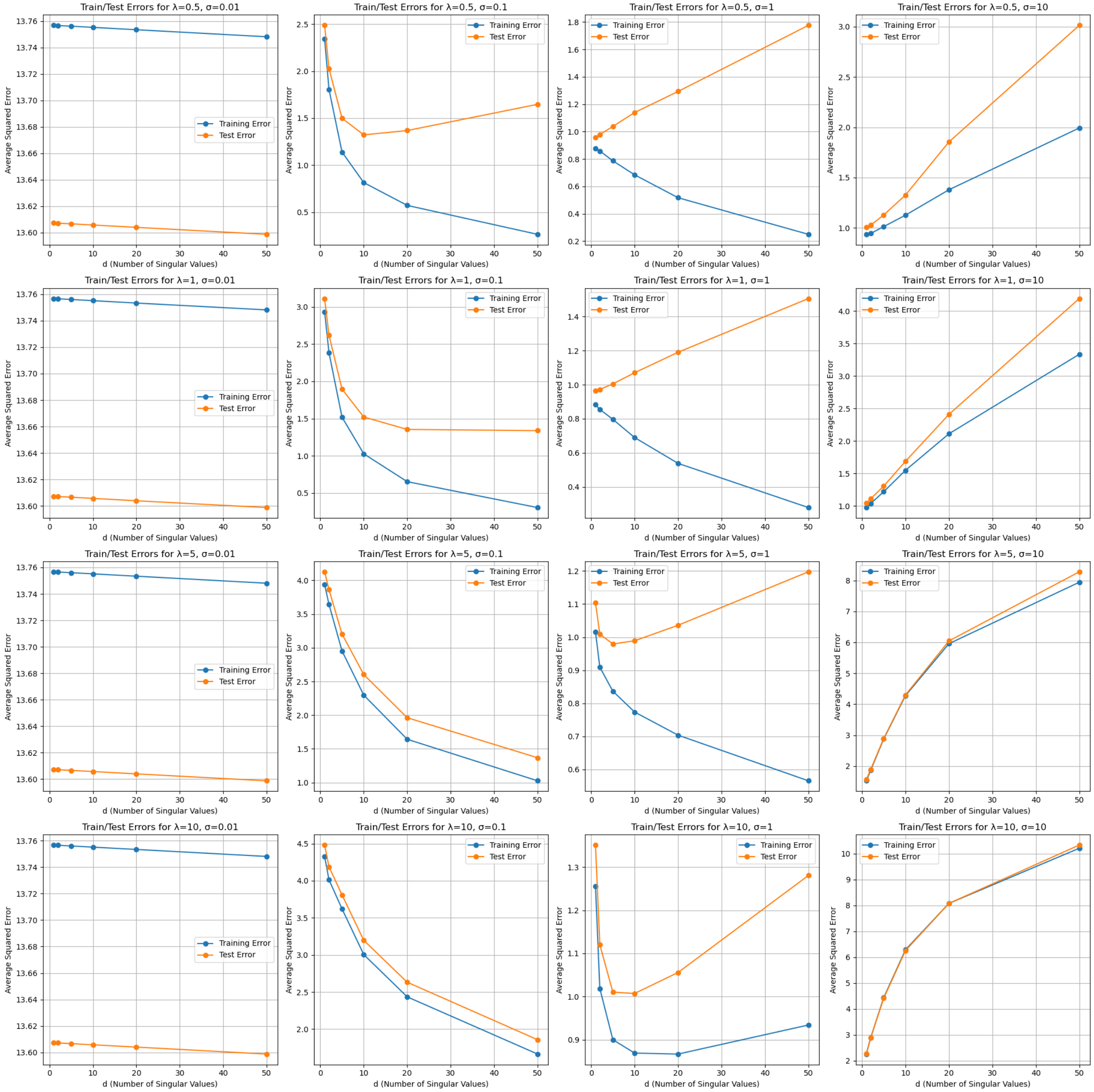
Test Error (MSE) for the estimator $\hat{R}$: 1.063564200567445

b



c




Train/Test Errors for λ=0.5, σ=0.01

Train/Test Errors for λ=0.5, σ=0.1

Train/Test Errors for λ=0.5, σ=1

Train/Test Errors for λ=0.5, σ=10

Train/Test Errors for λ=1, σ=0.01

Train/Test Errors for λ=1, σ=0.1

Train/Test Errors for λ=1, σ=1

Train/Test Errors for λ=1, σ=10

Train/Test Errors for λ=5, σ=0.01

Train/Test Errors for λ=5, σ=0.1

Train/Test Errors for λ=5, σ=1

Train/Test Errors for λ=5, σ=10

Train/Test Errors for λ=10, σ=0.01

Train/Test Errors for λ=10, σ=0.1

Train/Test Errors for λ=10, σ=1

Train/Test Errors for λ=10, σ=10

```
import csv
import numpy as np
from scipy.sparse.linalg import svds
import matplotlib.pyplot as plt
import torch
data = []
with open('u.data') as csvfile:
    spamreader = csv.reader(csvfile, delimiter='\t')
    for row in spamreader:
        data.append([int(row[0])-1, int(row[1])-1, int(row[2])])
data = np.array(data)

num_observations = len(data)  # num_observations = 100,000
num_users = max(data[:,0])+1  # num_users = 943, indexed 0,...,942
num_items = max(data[:,1])+1  # num_items = 1682 indexed 0,...,1681

np.random.seed(1)
num_train = int(0.8*num_observations)
```

```
perm = np.random.permutation(data.shape[0])
train = data[perm[0:num_train],:]
test = data[perm[num_train::],:]

print(f"Successfully loaded 100K MovieLens dataset with",
      f"{len(train)} training samples and {len(test)} test samples")

# Your code goes here
# Compute estimate
print(np.max(train[:, 1]))
# Calculate average rating for each movie
movie_ratings_sum = np.zeros(num_items)
movie_ratings_count = np.zeros(num_items)

for user, movie, rating in train:
    movie_ratings_sum[movie] += rating
    movie_ratings_count[movie] += 1

# Avoid division by zero
movie_ratings_count[movie_ratings_count == 0] = 1

average_movie_ratings = movie_ratings_sum / movie_ratings_count

# Construct rank-one matrix R_hat
R_hat = np.tile(average_movie_ratings, (num_users, 1))
# Evaluate test error

test_error = 0
for user, movie, actual_rating in test:
    predicted_rating = R_hat[user, movie]
    test_error += (predicted_rating - actual_rating) ** 2

test_error /= len(test)

print("Test Error (MSE) for the estimator R_hat:", test_error)

# Your code goes here
# Create the matrix R twiddle (\widetilde{R}).
R_tilde = np.zeros((num_users, num_items))
for user, movie, rating in train:
    R_tilde[user, movie] = rating

  # Your code goes here
def construct_estimator(d, r_twiddle):
    U, s, Vt = svds(r_twiddle, k = d)
    R_hat_d = U @ np.diag(s) @ Vt

    return R_hat_d


def get_error(d, r_twiddle, dataset):
    R_hat_d = construct_estimator(d, r_twiddle)
    error = 0
    for user, movie, actual_rating in dataset:
        predicted_rating = R_hat_d[user, movie]
        error += (predicted_rating - actual_rating) ** 2
    return error / len(dataset)

# Your code goes here
# Evaluate train and test error for: d = 1, 2, 5, 10, 20, 50.

d_values = [1, 2, 5, 10, 20, 50]
usv_train_errors = []
usv_test_errors = []
for d in d_values:
    usv_train_error=get_error(d, R_tilde, train)
    usv_test_error=get_error(d, R_tilde, test)
    usv_train_errors.append(usv_train_error)
    usv_test_errors.append(usv_test_error)

# Plot both train and test error as a function of d on the same plot.


plt.figure(figsize=(10, 6))
plt.plot(d_values, usv_train_errors, label='Training Error', marker='o')
plt.plot(d_values, usv_test_errors, label='Test Error', marker='o')
plt.xlabel('d (Number of Singular Values)')
plt.ylabel('Average Squared Error')
plt.title('Training and Test Errors vs. Rank of Approximation')
plt.legend()
plt.grid(True)
plt.show()

# Your code goes here
def closed_form_u(V, U, l, R_tilde):
  print(V.shape, U.shape)
  for i in range(U.shape[0]):
    R_i_tile = R_tilde[i]
    indices = R_i_tile > 0
    V_j = V[indices, :]
    R_i = R_i_tile[indices]
    A = V_j.T @ V_j + l * np.eye(V_j.shape[1])
    b = (V_j.T @ R_i).reshape(-1, 1)
    U[i,:] = np.linalg.solve(A, b).flatten()
  return U

def closed_form_v(V, U, l, R_tilde):
  for j in range(V.shape[0]):
    R_j_tile = R_tilde[:,j]
    indices = R_j_tile > 0
    U_i = U[indices, :]
    R_j = R_j_tile[indices]
    A = U_i.T @ U_i + l * np.eye(U_i.shape[1])
    b = (U_i.T @ R_j).reshape(-1, 1)
    V[j,:] = np.linalg.solve(A, b).flatten()
  return V


def construct_alternating_estimator(
    d, r_twiddle, l=0.0, delta=1e-2, sigma=0.1, U=None, V=None
):
  old_U, old_V = np.zeros((num_users, d)), np.zeros((num_items, d))
  if U is None:
    U = np.random.rand(num_users, d) * sigma
  if V is None:
    V = np.random.rand(num_items, d) * sigma
  while (np.max(np.abs(V - old_V)) > delta and np.max(np.abs(U - old_U)) > delta):
    old_U, old_V = U, V
    U = closed_form_u(V, U, l, r_twiddle)
    V = closed_form_v(V, U, l, r_twiddle)
  return U, V

def calc_uv_error(dataset, U, V):
    user = dataset[:,0]
    item = dataset[:,1]
    score = dataset[:,2]
    # print(U.shape, V.shape)
    pred = np.einsum('ij,ij->i', U[user], V[item])
    mse_error = np.mean((score-pred) ** 2)
    return mse_error

from itertools import product
d_vals = [1, 2, 5, 10, 20, 50]
lambdas = [0.5, 1, 5, 10]
sigmas = [0.01, 0.1, 1, 10]

# Prepare the plots
fig, axes = plt.subplots(4, 4, figsize=(20, 20)) # 4x4 grid for 16 combinations
axes = axes.flatten()

# Iterate over all combinations of lambdas and sigmas
for index, (lambda_val, sigma_val) in enumerate(product(lambdas, sigmas)):
    uv_train_errors = []
    uv_test_errors = []

    for d in d_vals:
        U, V = construct_alternating_estimator(d=d, r_twiddle=R_tilde, l=lambda_val, sigma=sigma_val)
        uv_train_errors.append(calc_uv_error(train, U, V))
        uv_test_errors.append(calc_uv_error(test, U, V))

    # Plot the train and test error for each combination
    ax = axes[index]
    ax.plot(d_vals, uv_train_errors, label='Training Error', marker='o')
    ax.plot(d_vals, uv_test_errors, label='Test Error', marker='o')
    ax.set_xlabel('d (Number of Singular Values)')
    ax.set_ylabel('Average Squared Error')
    ax.set_title(f'Train/Test Errors for lambda={lambda_val}, sigma={sigma_val}')
    ax.legend()
    ax.grid(True)

# Adjust layout
plt.tight_layout()
plt.show()
```