3.1

[87]:
```
import numpy as np
from tqdm import tqdm
import json
import matplotlib.pyplot as plt
from scipy.sparse.linalg import svds
```

[2]:
```
# Load the training data
train_data_path = "./train.txt"
train_data = np.loadtxt(train_data_path, delimiter=',')

# Load the test data
test_data_path = "./test.txt"
test_data = np.loadtxt(test_data_path, delimiter=',')

train_data.shape, test_data.shape
```

[2]:
```
((60000, 3), (60000, 3))
```

[3]:
```
# Define the matrix dimensions
n_users = 1000
n_movies = 500

# Initialize matrices with zeros
train_matrix = np.zeros((n_users, n_movies))
test_matrix = np.zeros((n_users, n_movies))

# Populate the train matrix
for i, j, s in train_data:
    train_matrix[int(i)-1, int(j)-1] = s

# Populate the test matrix
for i, j, s in test_data:
    test_matrix[int(i)-1, int(j)-1] = s

train_matrix.shape, test_matrix.shape
```

[3]:
```
((1000, 500), (1000, 500))
```

1 Q1

Here we implement the first estimator and evaluate our learnt vector representations by two metrics

[37]:
```
def simple_estimator(train_matrix, test_matrix):
    # Calculate the average rating for each movie in the training set
    movie_means = np.sum(train_matrix, axis=0) / np.count_nonzero(train_matrix,␣
 ↪axis=0)
    movie_means[np.isnan(movie_means)] = 0

    # Function to predict ratings
    def predict_ratings(matrix, movie_means):
        predictions = np.zeros_like(matrix)
        for j in range(matrix.shape[1]):
            predictions[:, j] = movie_means[j]
        return predictions

    # Calculate predicted ratings for train and test matrices
    train_predictions = predict_ratings(train_matrix, movie_means)
    test_predictions = predict_ratings(test_matrix, movie_means)

    # Function to calculate MSE and MAE
    def calculate_metrics(actual, predicted):
        mask = actual != 0
        mse = np.mean((predicted[mask] - actual[mask]) ** 2)
        mae = np.mean(np.abs(predicted[mask] - actual[mask]))
        return mse, mae

    # Calculate and print metrics for the training and test sets
    train_mse, train_mae = calculate_metrics(train_matrix, train_predictions)
    test_mse, test_mae = calculate_metrics(test_matrix, test_predictions)

    return train_mse, train_mae, test_mse, test_mae


train_mse, train_mae, test_mse, test_mae = simple_estimator(train_matrix,␣
 ↪test_matrix)
print(f"Training MSE: {train_mse:.3f}, Training MAE: {train_mae:.3f}")
print(f"Test MSE: {test_mse:.3f}, Test MAE: {test_mae:.3f}")
```

```
Training MSE: 0.559, Training MAE: 0.596
Test MSE: 0.570, Test MAE: 0.602
```