EE798R: Intelligent Pattern Recognition

Prof. Tushar Sandhan
sandhan@iitk.ac.in

Assignment - 2

Due date: 12 Nov, 2024 Weight: 15%
Due time: 11:59PM Submission: MookIT

---

1 Image Reconstruction and Classification: `CNN+NN+GMM` `[7.5%]`

Objective

In this problem, students are required to build a Convolutional Neural Network (CNN) in an encoder-decoder architecture for image reconstruction. They will then use the learned encoder as a feature extractor for classification with a feed-forward Neural Network (NN) and, finally, apply a Gaussian Mixture Model (GMM) to the NN's predictions to cluster the outputs for performance analysis.

Dataset: CIFAR-10

This dataset contains 60,000 32x32 color images across 10 different classes (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, trucks). It's widely used in image classification problems.

Part 1: CNN Encoder-Decoder for Image Reconstruction

- Implement a CNN in an **encoder-decoder** (autoencoder) architecture to reconstruct the CIFAR-10 images.
- The encoder should reduce the input image into a latent space (compressed feature representation), while the decoder reconstructs the image from the latent space.
- Train the encoder-decoder model to minimize the reconstruction loss (Mean Squared Error).

Submission for Part 1

- Save the **reconstructed images** for the test set in **NumPy array format (.npy)**. Each image should correspond to a test set image.
- **File Naming**: Name each reconstructed image as `image_000.npy`, `image_001.npy`, `image_002.npy`, etc., ensuring they align with the order of the CIFAR-10 test set.
- **Compression**: Combine all reconstructed `.npy` files into a single zip file named `reconstructed_images.zip`.

- **Directory Structure Example**:

```
submission/
|-- reconstructed_images.zip
|   |-- image_000.npy
|   |-- image_001.npy
|   |-- image_002.npy
|   `-- ...
```

- **Submission File**: `reconstructed_images.zip`

Autograding

The autograding script will extract the NumPy arrays from the zip file and compare them with the original CIFAR-10 test set images, calculating the **Mean Squared Error (MSE)** to evaluate reconstruction quality.

Part 2: Neural Network for Classification

- Use the features extracted from the encoder as input to a feed-forward neural network (NN) for image classification.
- Train the NN to classify the images into the 10 CIFAR-10 classes using the extracted features and evaluate the classification accuracy on the test set.

Submission for Part 2

- Save the **predicted class labels** for each test image in a **JSON** or **TXT** file.
- **File Naming**: Each entry should include the image name and its corresponding predicted class.
  - For JSON format: Use the format {`"image_name": class_label`}.
  - For TXT format: Use the format `image_name class_label`.
- **Example of JSON File**:

```
{
  "image_000": 3,
  "image_001": 1,
  "image_002": 9,
  ...
}
```

- **Example of TXT File**:

```
image_000 3
image_001 1
image_002 9
...
```

- **File Name**: `classification_predictions.json` or `classification_predictions.txt`
- **Directory Structure Example**:

```
submission/
|-- classification_predictions.json
```

- **Submission File**: `classification_predictions.json` or `classification_predictions.txt`

Autograding

The autograding script will load the prediction file and compare the predicted labels with the actual CIFAR-10 test labels, calculating the **classification accuracy**.

Part 3: Gaussian Mixture Model (GMM) Clustering

- Apply a **Gaussian Mixture Model (GMM)** on the output from the NN (before applying softmax or another activation function).
- Analyze the clusters formed by GMM and compare them with the true class labels using clustering performance metrics such as **Adjusted Rand Index (ARI)** or **Normalized Mutual Information (NMI)**.

Submission for Part 3

- Save the **cluster predictions** for each test image in a **JSON** or **TXT** file.
- **File Naming**: Each entry should include the image name and its corresponding GMM cluster.
  - For JSON format: Use the format `{"image_name": cluster_id}`.
  - For TXT format: Use the format `image_name cluster_id`.
- **Example of JSON File**:

```
{
  "image_000": 2,
  "image_001": 7,
  "image_002": 1,
  ...
}
```

- **Example of TXT File**:

```
image_000 2
image_001 7
image_002 1
...
```

- **File Name**: `gmm_clusters.json` or `gmm_clusters.txt`
- **Directory Structure Example**:

```
submission/
|-- gmm_clusters.json
```

- **Submission File**: `gmm_clusters.json` or `gmm_clusters.txt`

Autograding

The autograding script will load the GMM cluster file and compare it to the true labels, calculating **ARI** and **NMI** metrics to evaluate clustering performance.

Final Submission Directory Structure

Ensure your final submission follows the structure below:

```
submission/
|-- reconstructed_images.zip
|-- classification_predictions.json (or .txt)
|-- gmm_clusters.json (or .txt)
```

Autograding Script

Below is a Python script for autograding the submissions based on the submitted files. This script assumes that the CIFAR-10 test set labels are available and that the submissions follow the specified formats.

```python
import numpy as np
import json
from sklearn.metrics import mean_squared_error, accuracy_score,
adjusted_rand_score, normalized_mutual_info_score
import zipfile
import os
import torch
import torchvision
import torchvision.transforms as transforms

```

```python
def load_cifar10_test_set():
    """
    Load the CIFAR-10 test dataset to retrieve true labels.
    """
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5),  # Mean for each channel
                             (0.5, 0.5, 0.5))  # Std for each channel
    ])
    test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                download=True, transform=transform)
    return test_dataset

# Part 1: Evaluate Image Reconstruction MSE
def evaluate_image_reconstruction(zip_file, test_set):
    mse_losses = []
    num_images = len(test_set)

    # Extract images from zip
    with zipfile.ZipFile(zip_file, 'r') as z:
        for i in range(num_images):
            image_name = f'image_{i:03}.npy'
            with z.open(image_name) as file:
                reconstructed_image = np.load(file)
                # Get the original image
                original_image, _ = test_set[i]
                # Denormalize the original image
                original_image = original_image.numpy().transpose(1, 2, 0)
                original_image = original_image * 0.5 + 0.5  # Undo normalization
                original_image = np.clip(original_image, 0, 1)
                # Compute MSE
                mse = mean_squared_error(original_image.flatten(), reconstructed_image.flatten())
                mse_losses.append(mse)

    avg_mse = np.mean(mse_losses)
    return avg_mse

# Part 2: Evaluate Classification Accuracy
def evaluate_classification_predictions(pred_file, test_set):
    if pred_file.endswith('.json'):
        with open(pred_file, 'r') as file:
            predictions = json.load(file)
    elif pred_file.endswith('.txt'):
        predictions = {}
        with open(pred_file, 'r') as file:
            for line in file:
                parts = line.strip().split()
                if len(parts) == 2:
                    image_name, cls = parts
                    predictions[image_name] = int(cls)
    else:
        raise ValueError("Unsupported file format for classification predictions.")

```

```python
    true_labels = [label for _, label in test_set]
    predicted_labels = [predictions[f'image_{i:03}'] for i in range(len(test_set))]

    acc = accuracy_score(true_labels, predicted_labels)
    return acc

# Part 3: Evaluate GMM Clustering
def evaluate_gmm_clusters(cluster_file, test_set):
    if cluster_file.endswith('.json'):
        with open(cluster_file, 'r') as file:
            clusters = json.load(file)
    elif cluster_file.endswith('.txt'):
        clusters = {}
        with open(cluster_file, 'r') as file:
            for line in file:
                parts = line.strip().split()
                if len(parts) == 2:
                    image_name, cluster = parts
                    clusters[image_name] = int(cluster)
    else:
        raise ValueError("Unsupported file format for GMM clusters.")

    true_labels = [label for _, label in test_set]
    predicted_clusters = [clusters[f'image_{i:03}'] for i in range(len(test_set))]

    ari = adjusted_rand_score(true_labels, predicted_clusters)
    nmi = normalized_mutual_info_score(true_labels, predicted_clusters)

    return ari, nmi

# Autograding script execution
if __name__ == "__main__":
    test_set = load_cifar10_test_set()

    # Define submission directory
    submission_dir = 'submission'

    # Part 1: Evaluate Image Reconstruction
    reconstructed_zip = os.path.join(submission_dir, 'reconstructed_images.zip')
    mse = evaluate_image_reconstruction(reconstructed_zip, test_set)
    print(f"Reconstruction MSE: {mse:.6f}")

    # Part 2: Evaluate Classification Accuracy
    classification_pred_json = os.path.join(submission_dir, 'classification_predictions.json')
    classification_pred_txt = os.path.join(submission_dir, 'classification_predictions.txt')
    if os.path.exists(classification_pred_json):
        classification_acc = evaluate_classification_predictions(classification_pred_json, test_set)
    elif os.path.exists(classification_pred_txt):
        classification_acc = evaluate_classification_predictions(classification_pred_txt, test_set)
    else:
```

```python
            raise FileNotFoundError("Classification predictions file not found."
    )
        print(f"Classification Accuracy: {classification_acc * 100:.2f}%")

        # Part 3: Evaluate GMM Clustering
        gmm_cluster_json = os.path.join(submission_dir, 'gmm_clusters.json')
        gmm_cluster_txt = os.path.join(submission_dir, 'gmm_clusters.txt')
        if os.path.exists(gmm_cluster_json):
            ari, nmi = evaluate_gmm_clusters(gmm_cluster_json, test_set)
        elif os.path.exists(gmm_cluster_txt):
            ari, nmi = evaluate_gmm_clusters(gmm_cluster_txt, test_set)
        else:
            raise FileNotFoundError("GMM clusters file not found.")
        print(f"Adjusted Rand Index (ARI): {ari:.4f}")
        print(f"Normalized Mutual Information (NMI): {nmi:.4f}")
```

Additional Information on Dataset Loading and Usage

To standardize the dataset loading process and minimize variability, please follow these guidelines when handling the CIFAR-10 dataset:

1. Use PyTorch's torchvision library to load the CIFAR-10 dataset.
2. Apply consistent transformations, such as normalization, to ensure uniformity across all parts.
3. Maintain the order of images when saving reconstructed images and predictions to ensure alignment with the test set during autograding.
4. Utilize the indexing system (e.g., `image_000.npy`, `image_001.npy`, etc.) to preserve order.

Code Snippet to Use for Loading the CIFAR-10 Dataset

```python
    import torch
    import torchvision
    import torchvision.transforms as transforms
    from torch.utils.data import DataLoader

    # Define transformations
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.5, 0.5, 0.5),  # Mean for each channel
                             (0.5, 0.5, 0.5))  # Std for each channel
    ])

    # Load CIFAR-10 training and test datasets
    train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                                 download=True, transform=
    transform)
    test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                                download=True, transform=
    transform)

    # Define DataLoaders
```

```
    batch_size = 128
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True
    , num_workers=2)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False,
     num_workers=2)
```

Expected Results

- The CNN encoder-decoder should effectively reconstruct CIFAR-10 images with minimal loss (target MSE < **0.02**).
- The NN classifier should achieve **70% or higher accuracy** on the CIFAR-10 test set.
- GMM clustering should align reasonably with the original labels when evaluated using ARI or NMI, aiming mainly for an ARI score **above 0.5**.

2 Explicit Feedback-Based Recommender System: `Collaborative Filtering + Matrix Factorization + Autoencoders` [7.5%]

Objective

In this problem, students will implement three different techniques for building a recommender system using **explicit feedback** data: Collaborative Filtering, Matrix Factorization, and Autoencoders. The goal is to predict user ratings for unseen items based on the provided explicit feedback (e.g., 1–5 star ratings).

Dataset: MovieLens 100K Dataset

This dataset contains 100,000 ratings from 943 users on 1,682 movies, which will be used to train and test the recommender systems. It is widely used for collaborative filtering tasks.

Part 1: Collaborative Filtering

- Implement user-based **Collaborative Filtering** to predict missing ratings. Use similarity measures (e.g., cosine similarity) between users and predict ratings based on the weighted sum of ratings from the most similar users.
- Evaluate the model using **Root Mean Square Error (RMSE)** on the test set.

Submission for Part 1

- **Predicted Ratings**: Save the predicted ratings for the test set in a **CSV file** with the following columns:
  - `user_id`: ID of the user.
  - `item_id`: ID of the item.

  - `predicted_rating`: The predicted rating value.

- **File Name**: `collaborative_filtering_predictions.csv`
- **Example Format**:

```
user_id,item_id,predicted_rating
1,1193,4.5
1,661,3.0
2,914,2.5
...
```

- **Directory Structure Example**:

```
submission/
|-- collaborative_filtering_predictions.csv
```

Autograding

The autograding script will load this CSV file and calculate the **RMSE** by comparing the `predicted_rating` with the actual ratings in the test set.

Part 2: Matrix Factorization

- Implement **Matrix Factorization** using Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) to decompose the user-item matrix into latent factors.
- Predict the missing ratings using the learned latent factors and evaluate the model using **RMSE** on the test set.
- Regularize the matrix factorization model to prevent overfitting by incorporating an L_2 penalty term.

Submission for Part 2

- **Predicted Ratings**: Save the predicted ratings for the test set in a **CSV file** with the following columns:
  - `user_id`: ID of the user.
  - `item_id`: ID of the item.
  - `predicted_rating`: The predicted rating value.
- **File Name**: `matrix_factorization_predictions.csv`
- **Example Format**:

```
user_id,item_id,predicted_rating
1,1193,4.7
1,661,2.8
2,914,3.1
...
```

- **Directory Structure Example**:

```
submission/
|-- matrix_factorization_predictions.csv
```

Autograding

The autograding script will load this CSV file and calculate the **RMSE** by comparing the `predicted_rating` with the actual ratings in the test set.

Part 3: Autoencoders

- Implement an **Autoencoder**-based approach to reconstruct the user-item matrix by mapping it to a lower-dimensional latent space. Train the autoencoder using explicit feedback ratings to minimize reconstruction loss.
- Use the trained autoencoder to predict missing ratings and evaluate the performance using **RMSE**.

Submission for Part 3

- **Predicted Ratings**: Save the predicted ratings for the test set in a **CSV file** with the following columns:
  - `user_id`: ID of the user.
  - `item_id`: ID of the item.
  - `predicted_rating`: The predicted rating value.
- **File Name**: `autoencoder_predictions.csv`
- **Example Format**:

```
user_id,item_id,predicted_rating
1,1193,4.6
1,661,3.2
2,914,2.9
...
```

- **Directory Structure Example**:

```
submission/
|-- autoencoder_predictions.csv
```

Autograding

The autograding script will load this CSV file and calculate the **RMSE** by comparing the `predicted_rating` with the actual ratings in the test set.

Final Submission Directory Structure

Ensure your final submission follows the structure below:

```
submission/
|-- collaborative_filtering_predictions.csv
|-- matrix_factorization_predictions.csv
|-- autoencoder_predictions.csv
```

Autograding Script

Below is a Python script for autograding the submissions based on the submitted files. This script assumes that the train and test splits are saved as CSV files and that the predictions are provided in the specified CSV formats.

```python
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt
import os

def load_train_test(train_csv, test_csv):
    """
    Load the train and test datasets from CSV files.
    """
    train_df = pd.read_csv(train_csv)
    test_df = pd.read_csv(test_csv)
    return train_df, test_df

def evaluate_rmse(predictions_csv, test_df):
    """
    Evaluate RMSE for the predicted ratings.
    """
    predictions = pd.read_csv(predictions_csv)
    merged = test_df.merge(predictions, on=['user_id', 'item_id'])
    mse = mean_squared_error(merged['rating'], merged['predicted_rating'])
    rmse = sqrt(mse)
    return rmse

```

```python
    # Autograding script execution
    if __name__ == "__main__":
        # Paths to the submission files
        submission_dir = 'submission'
        collaborative_pred = os.path.join(submission_dir, 'collaborative_filtering_predictions.csv')
        matrix_factorization_pred = os.path.join(submission_dir, 'matrix_factorization_predictions.csv')
        autoencoder_pred = os.path.join(submission_dir, 'autoencoder_predictions.csv')

        # Paths to the saved train and test CSV files
        train_csv = 'train_split.csv'
        test_csv = 'test_split.csv'

        # Load train and test data
        train_df, test_df = load_train_test(train_csv, test_csv)

        # Evaluate Part 1 - Collaborative Filtering
        cf_rmse = evaluate_rmse(collaborative_pred, test_df)
        print(f"Collaborative Filtering RMSE: {cf_rmse:.4f}")

        # Evaluate Part 2 - Matrix Factorization
        mf_rmse = evaluate_rmse(matrix_factorization_pred, test_df)
        print(f"Matrix Factorization RMSE: {mf_rmse:.4f}")

        # Evaluate Part 3 - Autoencoders
        ae_rmse = evaluate_rmse(autoencoder_pred, test_df)
        print(f"Autoencoder RMSE: {ae_rmse:.4f}")
```

Notes

- Ensure that the `train_split.csv` and `test_split.csv` files are present in the working directory for the autograding script to function correctly.
- The autograding script calculates the **RMSE** for each part by comparing the predicted ratings with the actual ratings in the test set.

Additional Information on Dataset Loading and Usage

To standardize the dataset loading process and minimize variability, the following guidelines should be followed when handling the MovieLens 100K dataset:

1. **Loading the Dataset**:
    - Use pandas to load the dataset directly from the provided URL or from a local path if the dataset is uploaded.
    - Ensure that the dataset is read with the correct separator and column names.
2. **Creating Train/Test Splits**:
    - Perform an **80-20 split** for training and testing datasets.
    - Save the train and test splits as `train_split.csv` and `test_split.csv` respectively.

- Ensure that both CSV files contain the following columns: `user_id`, `item_id`, `rating`.

3. **Handling DataFrames**:
   - From the beginning, manipulate and process data using **pandas DataFrames** to ensure consistency across all parts.
   - This approach facilitates easier data handling and compatibility with various machine learning models.

Code Snippet to Use for Loading and Splitting the Dataset

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
column_names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv('https://files.grouplens.org/datasets/movielens/ml-100k/u.data',
                 sep='\t', names=column_names)

# Drop the timestamp as it's not needed
df = df.drop('timestamp', axis=1)

# Split into train and test sets (80% train, 20% test)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Save the splits to CSV files
train_df.to_csv('train_split.csv', index=False)
test_df.to_csv('test_split.csv', index=False)
```

Expected Results

- Collaborative Filtering should achieve an RMSE **less than 1.0** on the test set.
- Matrix Factorization methods should yield **RMSE less than 1.0** on the test set.
- Autoencoder models, depending on network depth and tuning, are expected to have an RMSE **less than 2.0** on the test set.

Libraries/Tools Required

To successfully complete this assignment, the following libraries and tools are required. Ensure they are installed in your working environment before starting:

- **Python 3.7+**: The programming language for all implementations and evaluation scripts.
- **NumPy**: For numerical operations and array handling.
- **PyTorch**: To implement Convolutional Neural Networks (CNNs), Neural Networks (NNs), and Autoencoders. This also includes the `torchvision` library for loading the CIFAR-10 dataset.

- **scikit-learn**: For Gaussian Mixture Models (GMMs), clustering metrics (e.g., ARI, NMI), and RMSE calculations.
- **pandas**: To handle and manipulate tabular data, particularly in the MovieLens dataset task.
- **tqdm**: To track the progress of loops and model training.
- **zipfile**: For compressing and extracting files, particularly for handling reconstructed images.
- **json**: For saving and loading classification and clustering predictions in JSON format.

You can install the necessary libraries using the following pip command:

```
pip install numpy torch torchvision scikit-learn pandas tqdm
```

■