

MPEG Microbiome Classification Challenge

Chigozie Nkwocha (Gozie)

Challenge Objective

- To use machine learning to predict where in the body a microbiome sample comes from, based on their metagenomic data, by classifying samples into stool, oral, skin, or nasal, using its 16S rRNA gene sequence.

Evaluation Metric

- Log Loss

Instructions on how to reproduce results

Folders

- The **data** folder contains the 8kmer counts for the train and test datasets, as well as their class labels (in **Train.csv**)
- The **models** folder contains saved autoencoder model (in .pth) and the standardscaler object used to scale input data before embedding extraction.
- The **fl_data** folder contains subfolders for each body site.
- The **preds** folder is where model predictions on the test data are stored.
- All files needed to run the scripts are saved in the **data** folder. 8-Kmer extraction and autoencoder embedding extraction have been saved; it is advised to not rerun the scripts to save time.

Python Scripts

Code Order

- For data preprocessing from extracting 8kmer sequences, run **mpeg_data_preprocessing.py**. Files will be saved **data/** folder
- For Feature extraction using autoencoders, run **autoencoder_script.py**

There's no defined order for running the centralised and federated learning models.

- For centralised learning output, run the **mpeg_central_model.py**
- For federated learning output, run the **mpeg_fed_learn_model.py** script
- The other scripts are (**fl_utils**, **visual_utils** and **modelling_utils**) are modules that the two modelling scripts depend on and need to be in the same folder as the centralised and federated learning scripts.

Note:

- The **mpeg_data_preprocessing.py** script was used to extract the kmer counts from the Fastq files on Kaggle. Please note that it is advised not to run as the folder paths were tailored as is on Kaggle. It can only be adapted by changing the folder paths to match yours, else, everything should work.

- Also, the `autoencoder_script.py` may be run to generate the autoencoder embeddings. However, I provided the saved weights in the `models` subfolder in `data` folder alongside a pickled file of the `standardscaler` object used to scale the input data before dimensionality reduction.

Hardware used

- Hardware as provided in Kaggle Notebooks. GPU P100 was used to train the centralised model
- GPU T4 X 2 was used for feature extraction using autoencoders.

Approach

The fastq files from the decompressed MPEG-G format files, hosted on Kaggle, by @Muham-madQasimShabbeer was used for this challenge.

Data Preprocessing

Data cleaning

- Removed incorrect bases (bases with Ns, if present)
- Removed bases with low quality scores (below 20) and reads with sequence lengths less than 50
- Created 8-kmer sequences, with no successive skips.
- Aggregated them into counts
- To save memory during storing, kmer counts were saved with unique 8-kmers as rows as sample IDs as columns.

Feature Engineering

- Kmer counts were transposed with samples as rows and kmers as columns.
- *Normalisation*: To prevent sample bias due to read length and depth, I normalised counts using the centred-log ratio method (CLR).
- *Dimensionality Reduction*: The high dimensional data from the kmer size selected required huge computational resource, hence I decided to go with developing an autoencoder from CLR normalised 8-kmer counts. The embedding size chosen was 64.
- Embeddings were further scaled and fed into the model network.

Centered Log-Ratio:

$$\text{CLR} = \log(1 + \text{counts}) - \text{mean}(\log(1 + \text{counts}))$$

Modelling

Kaggle Notebook was used (GPU P100 hardware).

Model Architecture

- A three-layer fully connected layer (PyTorch) with 192 neurons at the first layer with RMSProp optimiser and crossentropy loss with learning rate at 5e-4. Each dense layer had a 1-D batch-norm layer.

Federated Learning

Due to less time to learn the Application Programming Interface (API) of the library, I decided to implement it from scratch, with the help of Microsoft Copilot and also with the concept as described in the **Flower** package for federated learning.

Setup

- To maintain “privacy” of clients, data splitting was done to ensure that samples belonging to one subject are in one client alone.
- Each client is set up and instantiated with the parameters from the global model which stays on the server.
- Each client has similar configuration setting provided as Python dictionaries, with some allowed keys.
- The server where the global model resides aggregates parameter updates from each client in the network.
- Aggregation is done using ***Federation Averaging*** strategy by computing the weighted sum of their parameters in each round. The weights come from their respective sample sizes.
- After aggregation, the global model updates its parameters from here and sends back this update to all the clients for the next round.
- Because, scaling will be done by each client, a standard scaler object was temporarily fit on the training samples from the first client. After that, all clients send back their summary statistics (mean and variance) to the server, where they are aggregated using the same idea as described in weight parameter update (weighted sum).
- After this is done, the server updates the global model with this new information and then sends back to the clients which update their scaling parameters with this new information.
- Validation evaluation is done at both client and server levels, where each client evaluates their validation data with their respective parameters and sends back to the server, which further computes a weighted sum (validation sample size). At the server level, all clients (or some, based on what is in the server configuration), send their temporarily sends their validation data to the server for evaluation using the updated parameters of the global model.
- Finally, testing is done only at the server level using the updates of the global model. This is to ensure global model’s learning.
- Depending on the server configuration, not all clients are selected for training and validation. All clients for training and validation are randomly selected (some can be used in either one or two of the training and validation round).

NB. No submissions were made for the federated learning as it seemed I wouldn’t be on the top 10. Script was developed after the competition and test locally on an HP 840 computer hardware with 8GB RAM and 1TB SSD.

Other Approaches I tried

A. Kmers

- Generation of 6-kmer counts and applying dimensionality reduction strategies (PCA, SVD and Partial Least Squares).
- Generation of 10-kmer counts but was abandoned because a lot of computational resource was needed. Feature extraction took more than 12 hours.

B. Word embeddings

Another approach is using a DNA2VEC embedding model, which is a word2vec model developed from 3-8 kmer gene sequences.

C. Linear Model with dimensionality reduction

- Dimensionality reduction using SVD, PCA and PLS and using simple models (Logistic regression models implemented in Scikit-learn and PyTorch) and a neural network model. This turned out to be the best models on the leaderboard but for linear models only.

NB: Other approaches I would have tried

- Feature selection by removing normalised kmer counts using variance thresholding and selecting 8kmer sequences that meet a certain cutoff before modelling.