COMP4651 Final Project Report

Music Recommendation with Cloud Integration

Fall 2024 | Group 5

KWONG, Hoi Wa
hwkwong@connect.ust.hk

SI TOU, Tin Nok
tnsitou@connect.ust.hk

TONG, Tsun Man
tmtong@connect.ust.hk

Introduction

The purpose of our project is to leverage cloud computing to recommend the song to which a user might want to listen next, powered by machine learning (ML), based on various features related to the song, such as music genre, artists, and more. We have experimented with a multitude of methods, from model architecture to cloud computing deployment strategies, for optimal classification accuracy.

In this project, we make use of the KKBOX WSDM dataset [1], which features over 7 million entries of soundtrack playing history, together with attributes of the pieces, up till September 2017. Based on this dataset, we have engineered a PyTorch-based Feed-Forward Neural Network (FFNN) music genre classifier that consists of multiple fully connected (Dense) layers to identify correlations between the features of the song and its genre. We then deploy the classifier on an AWS EC2 instance for efficient data processing and model training. The code is available in our GitHub repository, where you can also find a demonstration video showcasing the project.

Background

ML-based algorithms have demonstrated remarkable performance in music genre classification. Neural network architectures such as FFNN and Long Short-Term Memory (LSTM) are commonly implemented models. Frameworks like PyTorch provide the flexibility and scalability required to develop and train these models efficiently.

Amazon Web Services (AWS) Elastic Compute Cloud (EC2) is a prominent cloud computing service that provides resizable compute capacity, enabling developers to deploy, manage, and scale applications seamlessly. EC2 supports a wide range of instance types optimized for various workloads, including those requiring high computational power and storage capabilities essential for large-scale ML tasks.

Related Work

Our work is inspired by relevant work in the open-source community, which aims at running neural networks on the cloud. One such example is Xing et al. [2] which extracts, from the mel-spectrogram of music tracks, the music latent factor vector using Weighted Matrix Factorization (WMF), which is subsequently passed to a Convolutional Neural Network (CNN) and LSTM model to distil latent spatial features of the source, combined with an embedding layer in parallel to predict users' preference for music based on the feature data from their playing history. While this solution is edge-cloud-based, which utilizes the resources of edge endpoints to operate the models, the speed of the classifier based on this framework is throttled by the complexity of the architecture, which yielded a decent accuracy at the sacrifice of compute efficiency.

Proposed Method

Leveraging the WSDM dataset, we decided to perform music preference prediction using a simple prediction model. The dataset will first go through a cleaning process to fill in empty data with a label of "`-1`" to indicate that the field is unavailable. Then the training dataset is split into training set and validation set with a 4:1 ratio, meaning 20% of the training data is used as the validation data. Below is the overview of the model architecture:

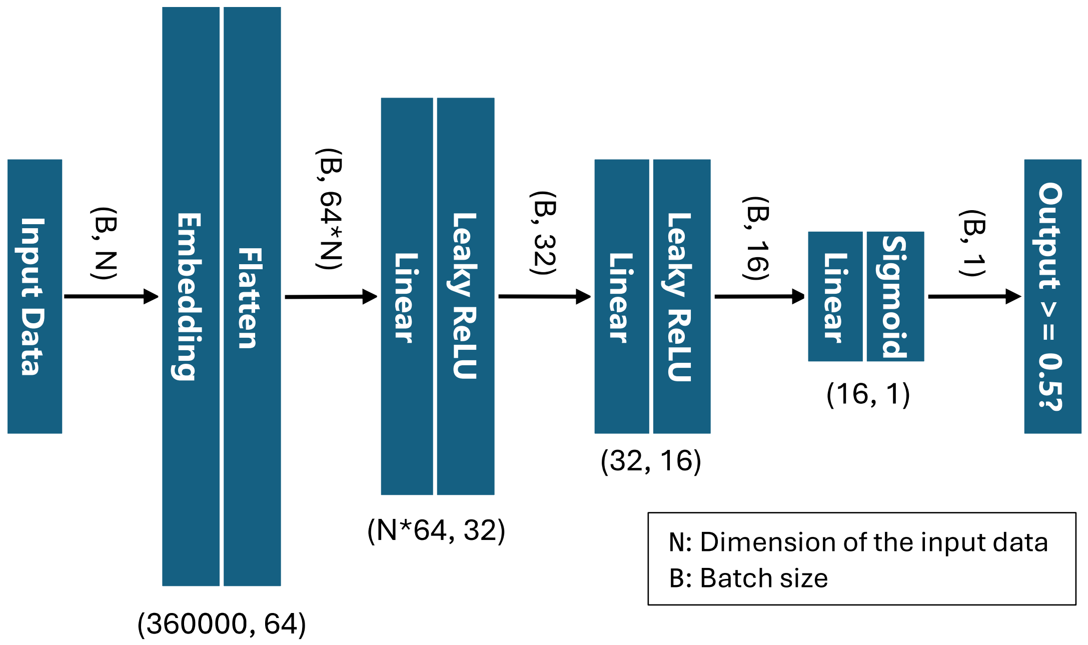


Figure 1. Overview of the proposed model.

After data pre-processing – further elaborated below – the input will be fed into the embedding layer and go through 3 linear layers of various neuron size and eventually output with a dimension of 1 with the sigmoid activation function. We determine that the user is likely to listen to the song again if the score exceeds 0.5, and vice versa.

<u>Additional model parameters</u> | Specified in main.py

- Optimizer: Adam
- Batch size: 800,000
- Learning rate: 0.1 (`1e-1`), 0.01 (`1e-2`), 0.001 (`1e-3`), 0.0001 (`1e-4`)
  | Learning rate scheduler: `torch.nn.ReduceLROnPlateau` (Patience: 3; Factor: 0.1)
- Number of epochs: 100 (Early stopping epochs: 10)
- Loss function: Binary Cross Entropy Loss (`torch.nn.BCELoss`)
  | Minimum improvement in validation loss to prevent early stopping: 0.01

Results

Upon testing, the model with learning rate `1e-1` does not converge at all, achieving the minimum accuracy of around 50%. Apart from this, the model with learning rate `1e-3` continued to grow gradually beyond the 20th epoch, in which the other models come to an early stop. It performed similarly as the model with learning rate `1e-2` despite beating the other models that grew more sharply with half the epochs, peaking at 70.15% of accuracy.

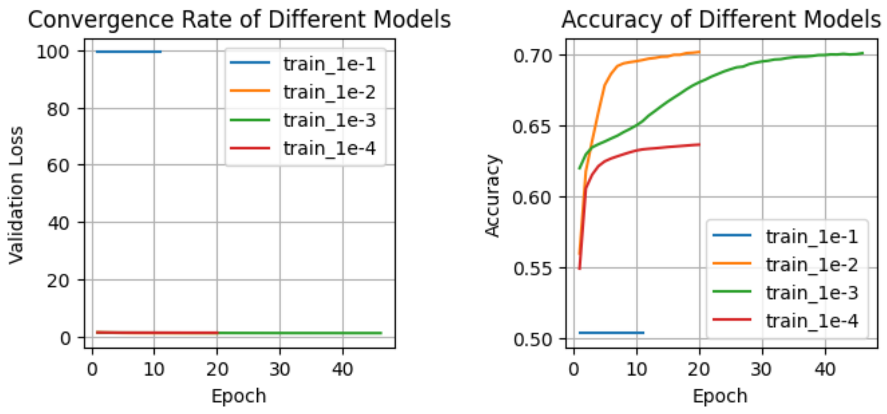


Figure 2. (a) Convergence rates and (b) accuracies of the model with different learning rates.

Dataset and Code Description

src/consolidate.py

Since the WSDM dataset is split into five .csv data files, including both the training and testing data, we use this file to store functions that fuse these data into unified Pandas DataFrames, one for training and one for testing, each of which consists of: track title, Int'l Standard Recording Code (ISRC) of the track (used to record the release year of the song and to identify duplicates), user information (including listen time tracked, genre preference, etc.), genre ID, and from where user decides to play the track (i.e. which interfaces do the user go through to begin playing the song). Several attributes from the dataset are then appended to these DataFrames in the file(s) below for future use.

src/transform.py

This file houses functions that process the previously obtained DataFrames by adding higher-level attributes to the data, as well as converting values in several fields into more machine-friendly formats:

- [Line 20] We added a field from the dataset to indicate whether the artist is featured by the platform as a trending one.
- [Lines 22-29] We added fields to indicate artists with more than one role in the production of the track, specifically if s/he is also the composer and/or the lyricist, as recorded on the dataset.
- [Lines 33-43; 45-55] For tracks with multiple kinds of genre or artists, we have separated them into 3 respective fields, the first two of which label the first two items of each (genre ID and artist name), and the last labels whether more are involved.
- [Lines 57-] Finally, we encode the remaining labels in the DataFrames using `sklearn.preprocessing.LabelEncoder` to facilitate the model training.

src/net.py

This file specifies the architecture of the FFNN model used in the project.

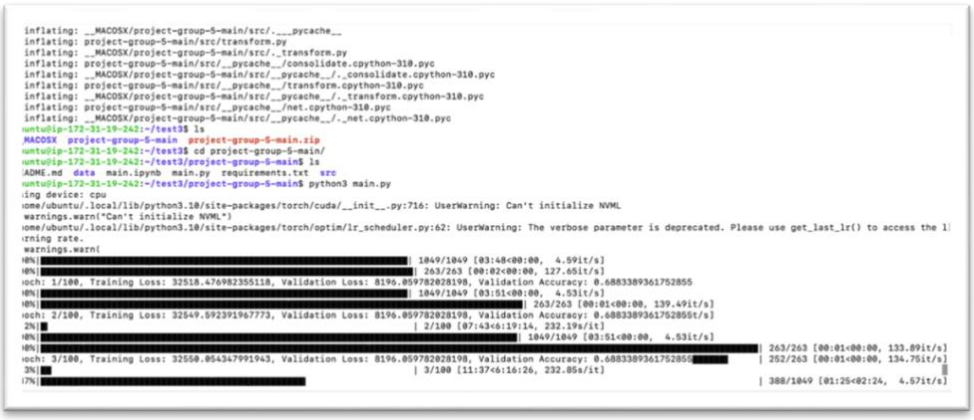
main.py, demo.py and aws_inference_service.py

At the heart of our project are these files, responsible for model training (main.py) and deployment (the other two). The latter two, powered by `flask`, are responsible for hooking up the models on AWS for the cloud integration part of this project, elaborated in the next section.

Cloud Deployment

Deploying a ML project using AWS EC2 involves leveraging its scalable and flexible virtual server infrastructure to meet diverse computational needs.

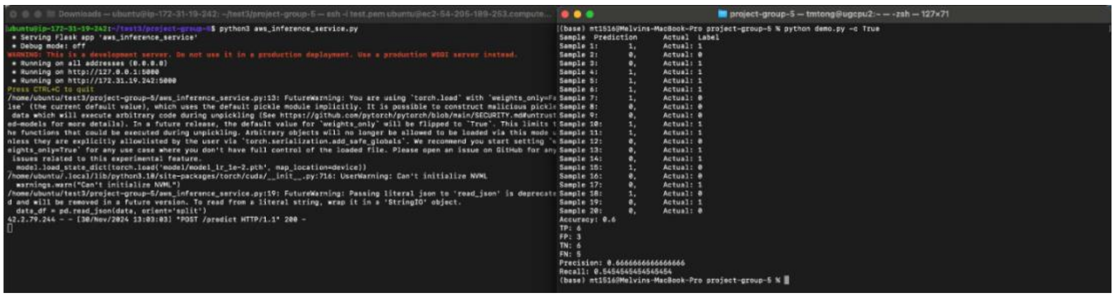
Deploying our machine learning project on AWS EC2 involved using the `t2.large` instance, which offers powerful CPUs ideal for training complex models. We began by launching the instance through the AWS Management Console, selecting Ubuntu as the operating system for its compatibility with ML frameworks. After setting up secure access via SSH and configuring the necessary security groups, we installed essential libraries such as PyTorch, NumPy, scikit-learn, etc. While the training process may be expedited using GPUs, given the relatively low complexity of the model, it is acceptable to be ran on a CPU-only instance.



```
inflating: __MACOSX/project-group-5-main/src/.___pycache__
inflating: project-group-5-main/src/transform.py
inflating: __MACOSX/project-group-5-main/src/._transform.py
inflating: project-group-5-main/src/__pycache__/consolidate.cpython-310.pyc
inflating: __MACOSX/project-group-5-main/src/__pycache__/._consolidate.cpython-310.pyc
inflating: project-group-5-main/src/__pycache__/transform.cpython-310.pyc
inflating: __MACOSX/project-group-5-main/src/__pycache__/._transform.cpython-310.pyc
inflating: project-group-5-main/src/__pycache__/net.cpython-310.pyc
inflating: __MACOSX/project-group-5-main/src/__pycache__/._net.cpython-310.pyc
untu@ip-172-31-19-242:~/test3$ ls
MACOSX  project-group-5-main  project-group-5-main.zip
untu@ip-172-31-19-242:~/test3$ cd project-group-5-main/
untu@ip-172-31-19-242:~/test3/project-group-5-main$ ls
ADME.md  data  main.ipynb  main.py  requirements.txt  src
untu@ip-172-31-19-242:~/test3/project-group-5-main$ python3 main.py
ing device: cpu
ome/ubuntu/.local/lib/python3.10/site-packages/torch/cuda/__init__.py:716: UserWarning: Can't initialize NVML
 warnings.warn("Can't initialize NVML")
ome/ubuntu/.local/lib/python3.10/site-packages/torch/optim/lr_scheduler.py:62: UserWarning: The verbose parameter is deprecated. Please use get_last_lr() to access the l
rning rate.
 warnings.warn(
0%|██████████████████████████████████████████| 1049/1049 [03:48<00:00,  4.59it/s]
0%|██████████████████████████████████████████| 263/263 [00:02<00:00, 127.65it/s]
och: 1/100, Training Loss: 32518.476982355118, Validation Loss: 8196.059782028198, Validation Accuracy: 0.6883389361752855
0%|██████████████████████████████████████████| 1049/1049 [03:51<00:00,  4.53it/s]
0%|████████████████████████████████████████████████████| 263/263 [00:01<00:00, 139.49it/s]
och: 2/100, Training Loss: 32549.592391967773, Validation Loss: 8196.059782028198, Validation Accuracy: 0.6883389361752855t/s]
2%|█                                         | 2/100 [07:43<6:19:14, 232.19s/it]
0%|████████████████████████████████████████████████████| 1049/1049 [03:51<00:00,  4.53it/s]
0%|██████████████████████████████████████████████████████████████████████████| 263/263 [00:01<00:00, 133.89it/s]
och: 3/100, Training Loss: 32550.054347991943, Validation Loss: 8196.059782028198, Validation Accuracy: 0.6883389361752855███   | 252/263 [00:01<00:00, 134.75it/s]
3%|█                                         | 3/100 [11:37<6:16:26, 232.85s/it]
7%|████████████████████████                                                     | 388/1049 [01:25<02:24,  4.57it/s]
```

Figure 3. Training process.

Throughout the model training process, we constantly monitor our machine learning models by analysing key performance metrics such as accuracy, precision and recall score using the validation dataset.



Downloads — ubuntu@ip-172-31-19-242: ~/test3/project-group-5 — ssh -i test.pem ubuntu@ec2-54-206-189-253.compute...
project-group-5 — tmtong@ugcpu2:~ — -zsh — 127×71

```
ubuntu@ip-172-31-19-242:~/test3/project-group-5$ python3 aws_inference_service.py
 * Serving Flask app 'aws_inference_service'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://172.31.19.242:5000
Press CTRL+C to quit
/home/ubuntu/test3/project-group-5/aws_inference_service.py:13: FutureWarning: You are using `torch.load` with `weights_only=Fa
lse` (the current default value), which uses the default pickle module implicitly. It is possible to construct malicious pickl
 data which will execute arbitrary code during unpickling (See https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrust
ed-models for more details). In a future release, the default value for `weights_only` will be flipped to `True`. This limits t
he functions that could be executed during unpickling. Arbitrary objects will no longer be allowed to be loaded via this mode u
nless they are explicitly allowlisted by the user via `torch.serialization.add_safe_globals`. We recommend you start setting `w
eights_only=True` for any use case where you don't have full control of the loaded file. Please open an issue on GitHub for any
 issues related to this experimental feature.
  model.load_state_dict(torch.load('model/model_lr_1e-2.pth', map_location=device))
/home/ubuntu/.local/lib/python3.10/site-packages/torch/cuda/__init__.py:716: UserWarning: Can't initialize NVML
  warnings.warn("Can't initialize NVML")
/home/ubuntu/test3/project-group-5/aws_inference_service.py:19: FutureWarning: Passing literal json to 'read_json' is deprecate
d and will be removed in a future version. To read from a literal string, wrap it in a 'StringIO' object.
  data_df = pd.read_json(data, orient='split')
42.2.79.244 - - [30/Nov/2024 13:03:03] "POST /predict HTTP/1.1" 200 -
```

```
(base) mt1516@Melvins-MacBook-Pro project-group-5 % python demo.py -c True
Sample  Prediction      Actual  Label
Sample 1:       1,      Actual: 1
Sample 2:       0,      Actual: 0
Sample 3:       0,      Actual: 1
Sample 4:       1,      Actual: 1
Sample 5:       1,      Actual: 1
Sample 6:       1,      Actual: 1
Sample 7:       1,      Actual: 0
Sample 8:       0,      Actual: 0
Sample 9:       0,      Actual: 0
Sample 10:      1,      Actual: 1
Sample 11:      1,      Actual: 1
Sample 12:      0,      Actual: 0
Sample 13:      0,      Actual: 1
Sample 14:      0,      Actual: 1
Sample 15:      1,      Actual: 0
Sample 16:      0,      Actual: 0
Sample 17:      0,      Actual: 1
Sample 18:      1,      Actual: 0
Sample 19:      0,      Actual: 1
Sample 20:      0,      Actual: 0
Accuracy: 0.6
TP: 6
FP: 3
TN: 6
FN: 5
Precision: 0.6666666666666666
Recall: 0.5454545454545454
(base) mt1516@Melvins-MacBook-Pro project-group-5 %
```

Figure 4. Demonstration of the Flask application running on AWS EC2 service and the batch processing function to submit the job locally.

After the training process, we have developed a simple flask-based API application to handle requests on the EC2 instance. As demonstrated in Figure 4, the API will respond to users' POST request to batch process the input data. "`demo.py`" includes a subset of 20 entries of the validation set just for demonstration purposes.

Discussion

- Architecture of classification model: We have come across various machine learning architectures that can effectively extract relevant features from the dataset and perform genre classification, such as FFNN and LSTM models. We have decided to utilise the former for its simplicity and effectiveness in task accomplishment with reasonable demand for computational resources.
- AWS EC2 and ECS: Having experimented with both deployment methods, we have encountered challenges in running our model on an ECS instance owing to the limited permission of the student account. Hence, we chose to deploy our model using EC2 due to its reliability and flexibility in scaling.

Future work

- Edge computing: By leveraging edge computing, a more lightweight version of the recommendation model can be enabled to more users using commodity devices, reducing latency and allowing the system to operate even if the user's device has limited connectivity or NPU performance. This could involve techniques like model compression, quantization, or federated learning to optimize the model for deployment on edge devices.
- Serverless model: Serverless architectures, such as those provided by cloud functions or serverless containers, can offer benefits like automatic scaling, reduced infrastructure management, etc. This could be particularly useful for handling the variable workloads and resource requirements of the music recommendation system, allowing it to scale up or down as needed without the overhead of managing a traditional server infrastructure.

Conclusion

Having experimented with cloud-powered, ML-based music preference prediction based on the KKBOX WSDM dataset, we have gained many useful insights from the project, from model fine-tuning to hands-on experience of model deployment on AWS EC2. While we have successfully compacted the model to fit on the `t2.large` instance, aside from further optimizing resource usage, harnessing more powerful models can slash the training time and expedite experimentation of various architectures and model sizes.

References

[1] KKBOX, “WSDM - KKBox's Music Recommendation Challenge,” 2017. [Online] Available: https://www.kaggle.com/competitions/kkbox-music-recommendation-challenge/data

[2] W. Xing, A. Slowik and J. Peter, “Edge-cloud computing oriented large-scale online music education mechanism driven by neural networks,” *J Cloud Comp,* no. 13, p. 55, 7 March 2024. doi: 10.1186/s13677-023-00555-y

Appendix 1. Genre ID mappings

While KKBOX did not reveal the meanings of genre IDs in the WSDM dataset, the genre information embedded in these data can be decoded by comparing the genre of the track entries with the same genre ID. Below are the mappings of genre information used in the project that cover over 95% of entries in the dataset:

ID	139	359	437	444	451	458	465	726
Genre	R&B	Soul	J-pop	K-pop	Mandopop	Pop	Rock	Rhymes
ID	786	921	958	1209	1609	1616	2022	2122
Genre	Folk	Score	Classical	Hip-hop	Electronic	House	Metal	Jazz