Experiment Design, Group 04 – Reproducibility

Option 2: Prediction of music genre across different taxonomies

Helmuth Breitenfellner
e8725866@student.tuwien.ac.at

László Király
e9227679@student.tuwien.ac.at

Gerald Weber
e0125536@student.tuwien.ac.at

Abstract

Based on the paper *"MediaEval 2018 AcousticBrainz Genre Task: A Baseline Combining Deep Feature Embeddings Across Datasets"* by Oramas et al., we were working on reproducing their results on predicting genres from different taxonomies using stacked neural networks.
As documented in this paper we were only partially successful in a full reproduction, mainly due to missing information in the paper and the accompanying repository and technical issues related with the vast amount of data to be processed.
This reproducability exercise was performed by Group 04 during the lecture *188.992 Experiment Design for Data Science*.

A) Task Description

The original paper describes an approach of predicting music genres as a "baseline approach" for the *MediaEval 2018 AcousticBrainz Genre Task*.
The conference provides a website for the classification task[1] which describes the task, the schedule and the dataset on a subpage[2] in detail.
Four classification sources ("allmusic", "discogs", "tagtraum" and "lastfm") have genres identified for audio files, using four different taxonomies of genres. While the audio files themselves are not available, there is a JSON file with precomputed audio features extracted using Essentia[3].
Using these JSON files and one-hot encoding for the categorical values contained, the authors have extracted a set of 2669 features which were then used to train a neural network per classification source. Each network has one 256-dimensional hidden layer and the output layer, corresponding in dimensionality with the genres used by the classification source.
As a follow-up step the authors have then stacked the hidden layer representation of songs based on the four networks into one 1024-dimensional feature vector, which is then used to predict the genre per classification source.
As a preparational step the training data was split into 80%/20% "train-train" and "train-test" data by the authors, used for validation during the training step.

(1) Data Sets

We downloaded the datasets provided by the authors via zenodo, in addition to the "allmusic" dataset from TUWEL (which is under a more restricted license and therefore not publicly available).
The ground truth is provided for 4019812 entries (most songs are classified by more than one classification source) as TSV-files.
Originally we also downloaded the test dataset. However since there is no "ground truth" available for them, predicting their genres without being able to validate the correctness is of no value for us. Therefore we removed them later on.
Overall we had to download data in the volume of more than 100GB (uncompressed), split into 1772307 single files in JSON format.
We have then randomly split the training data set into 80% "train-train" and 20% "train-test" data for the following tasks.

(2) Feature Extraction

The first step, which is extracting the 2669 features from the provided JSON files, is unfortunately not documented. We have tried contacting the main author via two email addresses, however we did not receive any response. This is very unfortunate as the features used presumably have a great impact on the final results.
In our attempts to recreate these features we were somehow lucky and found a repository[4] which was created just a few weeks ago, containing an attempt at creating the CSV files as needed.
This third-party repository is dealing with the 2019 iteration of the original paper. It was a good starting point, however a lot of work was required to achieve preprocessed data which is compatible with the other, provided code by the authors of the paper.
It did a good job at creating *almost* the correct number of features (2668 instead of 2669 as in the original paper), however one of the features has a constant value of 1. Also the encoding of the ground truth had some mistakes, and we had to implement a one-hot encoding to match the networks.
We ended up with adding one more feature, the *median* of the *beats_loudness*. It is missing in some observations and we are filling it with the mean for the other observations in the training dataset.
We have also ignored the fact that one of the features has a constant value, hoping that the impact of one feature, given the vast amount of features, will be neglectable. Again, with no documentation of the 2669 features chosen from the original paper it is difficult to know whether this is corresponding to the original implementation or not.
Another issue with the third-party code was that it performed the *z*-scaling using the mean and standard deviation estimated from *all* observations, including data from the validation set. We changed this to only use mean and standard deviation estimates derived from the "train-train" dataset, to avoid information leakage from validation data into the model.
When running the conversion the runtime of this step itself was a practical issue. Performing the conversation took more than 48 hours on the systems available to us. For development purposes we have therefore used a stripped-down version of the data. Still,

[1]https://multimediaeval.github.io/2018-AcousticBrainz-Genre-Task/, seen on 2020-01-30
[2]https://multimediaeval.github.io/2018-AcousticBrainz-Genre-Task/data/, seen on 2020-01-30
[3]https://essentia.upf.edu/, seen on 2020-01-30
[4]https://github.com/nikuya3/acousticbrainz-mediaeval-baseline, seen on 2020-01-30

when then applying the code on the full dataset new issues popped up, like missing values and their treatment. Overall, this "simple" step of feature extraction into CSV format took us multiple weeks to get it correct.

(3) Format Conversion

As a next step the original code was converting the data from CSV format into sets of files in HDF5 format, `numpy` persisted files and again TSV files for indexes.
This code was working as provided without any major issue, except that it was necessary using Python version 2.

(4) Learning the Subtask 1 Network

The code for learning the first network is based on Keras with a Theano backend.
Running the code on CPU only turned out to be completely infeasible.
Making Theano run on GPU is however not an easy task either. Development on Theano has stopped in 2017[5], and using GPU acceleration is only possible with outdated libraries and operating system drivers.
We therefore had to abandon the idea of using the existing code for the neural network. Since the network is rather simple in nature, reimplementing it in another platform turned out to be super simple. We used Pytorch where we have most experience in achieving GPU acceleration.
The result of subtask 1 are 4 models, each trained on the individual datasets (allmusic, discogs, lastfm, tagtraum) with ROC/AUC scores on the validation data.

(5) Learning the Subtask 2 Network

Our assumption is that following steps have been implemented in subtask 2:

- for each trained model
  - make predictions with all tracks from each dataset
  - save all track activations of the hidden layer for each dataset
  - use the track activations of the hidden layer from each dataset as input to train a new model

Example for allmusic:

- We train all 4 models as described in subtask 1
- We then predict allmusic dataset on all 4(!) models and save the activations of the hidden layer
- We the concatenate the 4 activations to one input to a new model which is trained on allmusic again, with an output layer to predict allmusic's genres.

Similar as for subtask 1, we were forced to re-implement the steps, as running GPU-accelerated Theano was not possible for us. We compared the number of trainable parameters of our implementation of the networks with the number in the original code and ensured that the numbers are equal.
For the ROC/AUC we first had an implementation which worked with Pytorch 1.4 and MacOS, but with pytorch 1.1 and Docker/Linux we faced the problem of 0 values for this metrics. Fixing this now broke the ROC/AUC metrics when running on Pytorch 1.4 and MacOS. Since we used Linux for the complete dataset we did not further investigate this curious difference.
The models for subtask 1 can be created by executing `run_subtask1.sh`, the activations by executing `prepare_subtask2.sh` and the final subtask 2 models by `run_subtask2.sh`.
The activations that we created with `prepare_subtask2.py` produced output which we were able to feed into the original code's with `prepare_subtask2` (not GPU-accelerated) and into our `train-subtask2.py`.

(6) Datasets

With the code from Section **??** we provided the dataset according to the script and started preprocessing, which consists of 3 steps:

- creating CSV feature file and a genres file per dataset
- calculating the mean and standard deviation of each dataset
- scaling each dataset

The output of the steps is written into *processed* folder. Even though the repository is very new and seems to handle the missing preprocessing step, an error occured in step 3:

```
Preprocessing train mode of allmusic dataset
Preprocessing validation mode of allmusic dataset
Preprocessing train mode of tagtraum dataset
Preprocessing validation mode of tagtraum dataset
Preprocessing train mode of discogs dataset
Preprocessing validation mode of discogs dataset
Preprocessing train mode of lastfm dataset
Preprocessing validation mode of lastfm dataset
Finished first preprocessing pass
Calculated means and standard deviations
Scale preprocessed datasets
Scaling processed/train/allmusic
Traceback (most recent call last):
...
ValueError: Unable to coerce to Series,
  length must be 2668: given 2669
```

Therefore *preprocessing.py* has been adapted. The idea is to use the preprocessing step of the second repository to generate the files for the data-preparation step of the original repository. Afterwards the training should have all required files to start (see figure 1).

(7) Preprocessing

- preprocessing.py python3
- participant_split_data.py
- create_h5.py python2 with h5py

Problems:

- TypeError: No conversion path for dtype: dtype('<U38') https://github.com/h5p
  -> solved with using python2

(8) Train

all python3

- python run_experiments.py genre_allmusic
- python run_experiments.py genres_discogs
- ...
- python run_experiments.py genres_allmusic_multimodal part of our Reproducibility run???

Problems:

- ValueError: Error when checking target: expected dense_5 to have shape (766,) but got array with shape (1,)

[5]https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ, seen on 2020-02-04

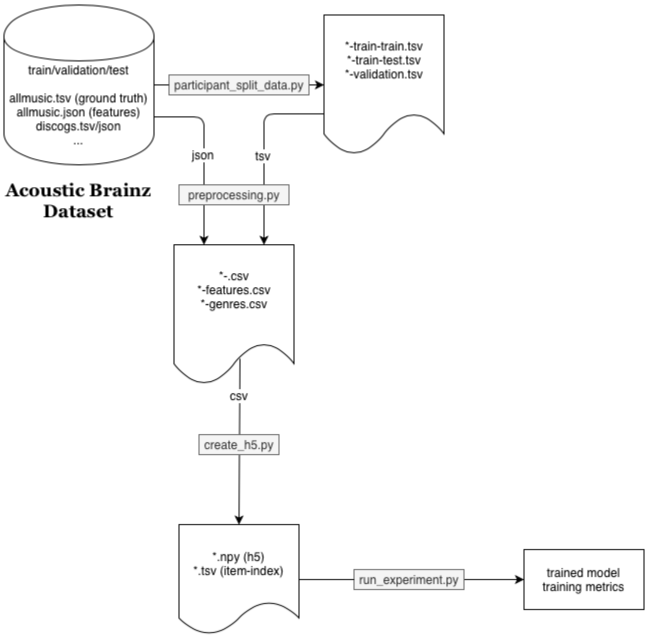

Figure 1: Preprocessing Data.

	AllMusic	Discogs	Lastfm	Tagtraum
Subtask 1			0.85281	0.82729
Subtask 2				

Table 1: ROC AUC on validation datasets

	AllMusic	Discogs	Lastfm	Tagtraum
Subtask 1	0.6476	0.7592	0.8276	0.8017
Subtask 2	0.8122	0.8863	0.9064	0.8874

Table 2: Original ROC AUC from paper

(9) GPU

Problems:

- Without GPU one epoch with only few 100 items takes hours -> reason was infinite loop due to too small(sic!) dataset in tartarus
- cuda on windows 10 fails: https://github.com/Theano/libgpuarray/issues/587

Workaround:
Train pytorch models with gpu.
Implemented in subtasks/train.py and subtasks/models.py
Problems:

- ValueError: Only one class present in y_true. ROC AUC score is not defined in that case.

(10) Sources

(11) Doing

B) Results

References