Inferred Hierarchical Classification for Deep Networks using Hierarchical Linkage Trees on Limited Input Spaces

Sierra Mangini [1] **Joshua Malcarne** [1] **Natalie McClain** [1] **Matthew Suyer** [1]

Abstract

This paper proposes an approach to improve the classification accuracy of fine-grain music genres by introducing inferred hierarchical information to a deep learning model. The proposed method uses metadata provided by Spotify's API and creates an "embedding" that represents songs using normalized features. To infer hierarchical relationships among genres, we create an Inferred Hierarchical Classification Tree (IHCT) through hierarchical clustering. The IHCT is used as input to a deep neural network, inspired by YOLO/YOLOv3 architecture, to classify the songs into genres. The paper contributes to the research community by introducing a novel approach to incorporating inferred hierarchical information into a deep learning model. We found it performed roughly equivalent to a simpler approach, but its results could prove useful for specific use cases.

1. Introduction

There exist instances where classification systems have larger and more fine-grained output spaces than there are readily available input spaces. One such example is music genres. Music genres have very nuanced classes, however, the most representative information such as audioforms and lyrics are not easy to acquire. In terms of Spotify, there exist 5238 recognized genres but no official way to get the audio files or lyrics. Instead, we have access to metadata such as runtime duration or featured artists. Luckily, Spotify creates and provides several audio features that numerically represent details about the songs such as 'acousticness', 'loudness', and 'tempo'.

Another common detail amongst these classification systems, including in the case of music genres, is that there may exist some official or unofficial hierarchy amongst the classes. Genres have sub-genres and super-genres, and these relations may be partially inferred in the available input space. We believe that by introducing inferred information about these hierarchies to a deep model we can increase the classification accuracy than a model of the same reduced input space without such inferences.

1.1. Research contributions

This paper contributes to the research community by introducing a novel approach to incorporating inferred hierarchical information into a deep learning model. Using this unique approach, we aim to improve the classification accuracy of music genres with fine-grained labels. Through this project, we hope to offer unique insight into the performance of our proposed approach, relative to established baselines and off-the-shelf models.

2. Related Work

Our proposed approach for this problem space is inspired by the YOLO/YOLOv3 architecture. YOLO, or "You Only Look Once", is a computer vision architecture that maps image features and associated probabilities spatially; the initial convolutional layers of the architecture extract features from a given image, whilst the subsequent fully connected layers classify the images' objects according to these features and their associated probabilities (Redmon et al., 2015). This method of unified detection enables significantly higher processing speed. YOLOv3 further builds upon this effectiveness by trading off a tiny bit of speed for a larger network and higher accuracy (Redmon & Farhadi, 2018). In our section 3, we are inspired indirectly by the concept of 'simultaneous' classifications. We classify multiple 'features' simultaneously, however, in our approach, we are classifying the ground-truth output at different levels of detail in a 'tree' format.

3. Proposed Method

We have available to us metadata provided by Spotify's API. With it, we create an 'embedding' to represent songs.

[1]Worcester Polytechnic Institute. Correspondence to: Sierra Mangini <smmangini@wpi.edu>, Joshua Malcarne <jrmalcarne@wpi.edu>, Natalie McClain <nmcclain@wpi.edu>, Matthew Suyer <mrsuyer@wpi.edu>.

CS/DS 541 Deep Learning Final Project, Worcester Polytechnic Institute. Copyright 2023 by the author(s).

It uses song duration, along with 9 of the 10 available 'audio features' and a song's popularity score. All of these are continuous numerical values that we normalize via Z-scoring. The normalized features are treated as embeddings that are 11 features long. This is the input being used across all our experiments, which is a relatively small input space. We hope that despite this small input space our proposed improvements will still result in increased accuracy.

3.1. The Inferred Hierarchical Classification Tree

Our methods are very reliant on the existence of an inferred hierarchy amongst the output classes, using the reduced input space. To do this, we create a representation of our classes with which to do hierarchical clustering. This would give us a tree such that at the first tier, T_1, we have 1 cluster containing N items (our N classes). At tier T_N we have N clusters containing 1 item each, 1 of each of our classes. At any $k; 1 \leq k \leq N$, there exists a tier T_k containing k clusters containing a constant total sum of N items across the clusters. We refer to this tree as an **Inferred Hierarchical Classification Tree** or **IHCT**. We refer to it as 'inferred' because we don't use any knowledge about any actual hierarchy that may or may not exist.[1] It is created from a limited input space and how close the classes are to each other within this space.

We denote a tier in the tree that contains k labels as T_k, while we denote the i'th tier from the top as T^i. This means that a tier that exists 4th from the top and that contains 16 clusters is denoted as T^4_{16}. We denote the maximum number of tiers of any given tree as T^I. The last tier in an *IHCT* will *always* contain all N classes as individual clusters, meaning that $T^I = T_N = T^I_N$.

In a completely 'natural' *IHCT*, $i = k$, where the 1st tier has 1 cluster, the 2nd tier has 2, and so on. Working with the natural tree in its entirety is not efficient. Instead, we are often concerned with some subset of tiers where we condense the tree to just this subset. In this case, we take cross-sectional cuts from a natural tree. For example, if we were to begin with an *IHCT* where $N = 12$, a valid subset of tiers we may be interested in is the tiers that are multiples of 3. In the case of such a subtree, we would have T^1_3, T^2_6, T^3_9, & T^4_{12}.

4. Experiment

4.1. IHCT Implementation

To create a realized version of our proposed *IHCT*, we first must create realized representations of our output classes.

[1]In terms of music genres, there could be manual labeling of sub/super genres however 5238 genres are too many for our timeframe. The concept will be discussed more in section 6.

We naively take the feature-wise mean of song embeddings, aggregated per genre. This gives us the 'average' song per genre. We then take these representations and pass them through hierarchical linkage functions.

We try a combination of **Complete Linkage, WPGMA, UPGMA,** and **Single Linkage**. We also experiment with using cosine similarity and Euclidean distance for the distance function in our linkages. This generates for us an *IHCT* as defined in section 3, allowing us to select any arbitrary subset of class clusters. We chose to train and test our model on the following subsets[2]:

$$N = 5283$$

$$S_1 = T^i_k; i \in [1..\lceil \log_2(N) \rceil]; k = \min(2^i, N)$$

$$S_2 = T^i_k; i \in [1..\lceil \log_2(N) \rceil]; k = \lceil N * \frac{1}{2}^{\lceil \log_2(N) \rceil - i} \rceil$$

4.2. Model Architectures

We used three different architectures: A 2-layer dense neural network, a series of progressively refined dense networks (or *StackNet*), and a GRU-based recurrent neural network.

4.2.1. Simple Dense Network

The 4-layer dense neural network served as our 'deep' baseline. We chose three dense layers of 256 hidden units, each followed by a 10% dropout, and a final dense layer of N units with softmax activation. It accepts a song embedding s and produces a one-hot encoding of length N to represent one of our N genres. In our case, $N = 5238$.

4.2.2. The 'Stack' Network

The progressive dense network, or as we call it the **Stack Network**/*StackNet*, works by training a simple dense network per tier of our tree. The first network, $\mathbf{net}_0$, accepts a song embedding s and outputs a prediction $\hat{T}^1$. We train this network on the ground-truth labels from T^1 from the generated *IHCT*. We then create $\mathbf{net}_1$ which accepts both s and $\hat{T}^1$ concatenated together into a single sample, and produces $\hat{T}^2$. This is trained on the ground-truth labels T^2 and again concatenated with both s and $\hat{T}^1$ creating $[s + \hat{T}^1 + \hat{T}^2]$.

We continue to 'stack' on additional networks up until $\mathbf{net}_{I-1}$ which predicts $\hat{T}^I_N$ at the I'th tier. In our case, $N = 5238$, but I varies depending on the subset of tiers selected.

[2]In simpler terms, we take the powers of 2, and recursive halving of N, but the math was too pretty to leave out.

$$s \rightarrow \hat{T}^1$$
$$[s + \hat{T}^1] \rightarrow \hat{T}^2$$
$$[s + \hat{T}^1 + \hat{T}^2] \rightarrow \hat{T}^3$$
$$[s + \hat{T}^1 + \hat{T}^2 + ... + \hat{T}^{I-1}] \rightarrow \hat{T}^I_N$$

This seems like the exact use case for a recurrent network[3] but it avoids a potential problem in our specific case. The number of classes being predicted at T^{j+1} will always be greater than T^j, meaning that both the input and output space *steadily grows*, along with the number of parameters, as we traverse down the *IHCT*. Deep learning frameworks require that RNNs have defined constant sequence lengths and sizes. The naive solution would be to pad our output space to N, however alongside a significant amount of wasted space we run the risk of running out of memory entirely, especially as the number of parameters grows. This is a frequent occurrence. The naive solution with no space wastage is the iterative growing of the models' input-output space as in the *StackNet*.

4.2.3. RECURRENT NETWORK

To circumvent the growing input-output space issue, we can do some data augmentation of our *IHCT*. Currently, we define each of the k clusters at T_k as a one hot that is k features long. This means at our final output space we have an output N features long (5238 to be specific). We can avoid this by instead referring to each cluster in the tree relative to its parent, and their parent's parent, and their parent's parent's parent, all the way to the root clusters at T^1. The tree isn't perfectly balanced, especially amongst certain linkage functions, however, the maximum number of children any one node in the tree has is far less than our N. Doing this reduces the peak length of our one-hots making padding all predictions once again feasible. We will denote the peak one-hot length as O.

In terms of the actual RNN architecture, we use a *bidirectional* ***GRU***, with a sequence length I and 256 hidden units. I of course being defined by the particular *IHCT* currently in use. The first time step predicts $\hat{T}^1$, the second time step predicts $\hat{T}^2$, and so on up to $\hat{T}^I_N$. We pass this forward to another two dense layers of shape $(I, 128)$ units and 10% dropout each, and a final (I, O) dense layer with softmax activation.

There are several clear advantages to this over the *StackNet*. The *StackNet* had to be trained *per tier* of the *IHCT*, and contains overall more parameters, while the RNN can all tier estimates at once in fewer parameters[4]. This is intuitive, as we have redefined our problem to be in terms of sequential classification. Alongside being more efficient, we found in most cases it performed better than the *StackNet* at classification as will be discussed in section 5.

[3]it is, which we will discuss in section 4.2.3 shortly

[4]Though not as few parameters as the simple dense network as will be seen in section 5.

5. Results

Our experiments showed that training using inferred hierarchical structure can in fact improve classification accuracy if only marginally, given the proper structure and linkage function are chosen.

5.1. Baseline

We used random guessing alongside a decision tree to develop a sense of baseline accuracy. With 5238 classes, if we were to randomly guess the class label every time, we would achieve an accuracy of 0.02%. The decision tree model utilized the song features as well as the genre label to partition the data based on entropy. Due to memory limitations, we were only able to train one tree (as opposed to a random forest) with a depth of 16, and we used an integer representation of the genre label as opposed to a one-hot encoding or some other embedding. The decision tree model was trained using a random sampling of 80% of the Spotify data we scraped, tuned on 10% of the data, and finally tested on the last 10% of the data. The model achieved an accuracy of 8.93% on the testing data.

5.2. Deep Models

We found our simple dense network achieved an accuracy of 13.3% with 30 epochs. This model's basis was a Tensorflow Keras sequential model, with dropout layers. *StackNet* found an accuracy of 10.2%, which was a significant improvement over our baselines but not as high as other models. The RNN was the best of all the models we trained, with a 13.5% accuracy, just barely beating out the simple dense network.

5.3. Overall

The maximum ground-truth accuracy can be seen as

Model	**Accuracy**
Random Guessing	0.02%
Decision Tree	8.93%
Simple Dense	13.3%
'Stack' Network	10.2%
RNN	13.5%

We found that the deep hierarchical RNN outperformed the other models ever so slightly. Ultimately, our proposed method was not a significant improvement over a simpler architecture, however, there are still points in its favor discussed in section 6.

6. Discussion

A noteworthy issue we ran into early on is the ratio of training samples to classes. Despite having 5238 different classes, we were able to collect only 200 examples each largely due to API rate limits and Spotify's TOS. Compared to the thousands of examples for 10 classes seen in MNIST, we have an underwhelming amount of information per class. This can be addressed by a larger dataset, which then introduces issues of resources. At approximately 635k training samples, we are already facing issues with memory usage. Increasing this by orders of ten would cause far more issues.

While our Hierarchical RNN came out on top, it only scraped by on fractions of a percentage point despite much more instrumentation involved. We believe that a limiting factor is the applicability of hierarchical linkages to the output classes. Our hierarchies are created very naively. Rather than on inferred hierarchies, our proposed architecture could still prove applicable to complex *known* hierarchies with defined parent-child relations.

7. Conclusions and Future Work

Our hierarchical approach was not significantly more accurate than other models, but both it and the naive deep network achieved similar levels of accuracy of around 13%. In the future using a larger dataset with more powerful computing could result in more accurate models, as we ran into memory issues. We could also incorporate more robust methods than just linkage clusters to represent relations. Perhaps in a more explicitly defined graph structure, recurrent Graph Convolutional Networks could be of use during future research.

References

Redmon, Joseph and Farhadi, Ali. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

Redmon, Joseph, Divvala, Santosh Kumar, Girshick, Ross B., and Farhadi, Ali. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.