Deep Learning for Music: Similarity Search and Beyond

Matthieu Gaston Michel Albert Devaux[1]

Distributed Information Systems Laboratory (LSIR), EPFL

Abstract

This paper is a study on the use of Machine Learning and Deep Learning to determine the similarity between different musics. The first part introduces the framework used and some concepts of Deep Learning that will be used in the algorithms. The second part talks about the problem caused by the topic and how the models should be to be efficient. The third part describes the algorithms used for trying to solve this problem. The fourth part is the analysis and testing of the model, followed by the conclusion to be drawn from them.

Keywords: Music Information Retrieval, Deep Learning

1. Introduction

We are currently at a time overflowed with music. Never before in history did we have so many musics easily accessible for us to listen. Take for example spotify : on only one application there are already tens of millions of songs available. If we were to listen to the first ten seconds of every of those musics, we would spend years not doing anything else. And more than the number, there is the diversity of musics : from the old styles, like gospel or symphony, to newer ones, like rap and rock. From subgenres that create splits, the way metal was split into a multitude of different styles, to the fusion of different genres that create convergence, like the creation of the county rock. In a world of such size and complexity, a question begins to emerge : what to listen to?

Historically, the first method used for advising music to people (whose field of research is called recommendation system) was content agnostic, meaning that the content of the music was not taken into account when doing the model, and is only based on what the users like or listen to. An example of this is the use by Spotify of collaborative filtering [1]. Those methods have advantages, it is a model that answers the problem of recommendation systems and it is way less complex that deep neural network solution, but it also has very big flows : because the method needs to know what people listen to before recommending them to the users, the musics that are not well-known or that are new have difficulties being put forward. By corollary, the musics that are often listen to, are more associated to other musics, so are more recommended, which creates a risk that the system proposes only popular songs. In a word, those model tend overfit on the dataset given.

Since then, there have been multiple recommendation systems developed using the content of the music to train and the result of the old models as labels [2]. But on this project, yet another method will be implemented : we will consider the problem of recommendation system for songs as a search engine that, given a music, returns the most similar musics in a given dataset. With this method, and given that the dataset used is not too biased, we would prevent the usual flaws of those models because the metadata of the music, or the people listening to it, would not influence which music is put forward. And even if it were to happen (for example because of the dataset used for the training), with a good similarity function, we would have a comparison of every songs, so we could get songs that are both similar enough so that it is a good suggestion, and different enough so that it gives unaccustomed results.

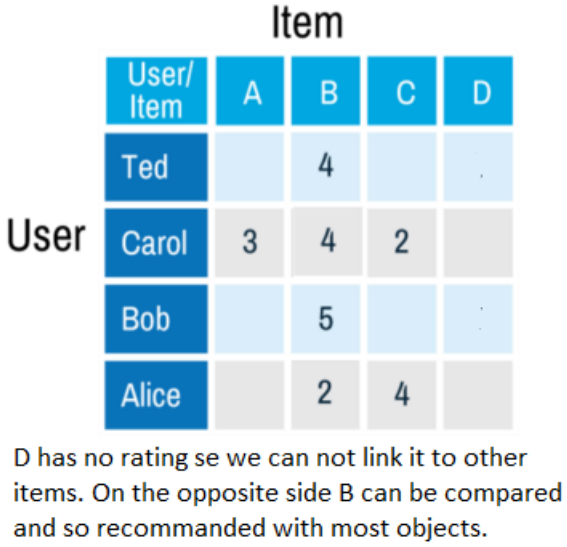


Figure 1: example of Collaborative Filtering limitation.

The aim of that project is to construct such a model, using machine learning and deep learning, to show that this similarity approach of the recommendation system for musics (and maybe also in a broader sense) is promising in this field. This deep

[1] Supervisors: Nguyen Thanh Tam, Karl Aberer

model will also be compared to non-learning models of similarity to show that neural networks could be an improvement in the field of Music Information Retrieval (MIR).

2. Overview

2.1. Background

2.1.1. Machine Learning

Machine Learning is a method that, given a function (called loss function) and a dataset, creates a model that optimize the function (we call it training) on the dataset in a way that this precision can be generalized to equivalent dataset. There are two reasons that a model can be bad : either it was not able to learn from the dataset on which it was trained on and it is called underfitting or while learning it was too tailored to the training dataset in a way that the model can not be generalized, it is called overfitting. The algorithm used for the model has what is called hyperparameters, they are parameters that creates the model, and so they can not be optimized directly by machine learning [3, 4, 5, 6].

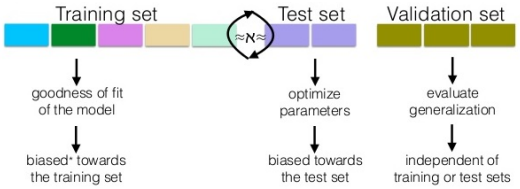


Figure 2: Schema of a split dataset.

To find the best model, a dataset must be split into three parts : the training dataset that it used by the model to train, the validation dataset that is used to evaluate how the model do on another dataset to prevent it from overfitting the training dataset, and the test dataset that is used to test a final model independently of the datasets use for training it. While making the model, datasets were split into : 50% for the training set, 25% for the evaluation set and 25% for the test set.

Neural network is a machine learning algorithm whose structure mimics a (very simplified) brain. It represented in three parts : the input layer where the input dataset is fed, the output layer which is the result of the network, and the hidden layer that are responsible for the changes between the input and output layer and which contains the variable to train. If a neural network has more than one hidden layer, it is called a deep learning neural network [7, 8, 9, 10].

In the example, the hidden layers are fully connected layers, because all the neurons of the hidden layers are connected to all the neurons of the layer before. If we consider the input of the full layer as a vector x, the output of the full layer is $\sigma(Ax + b)$ where A is a matrix, b a vector that can both be trained and σ is a function called activation function. A popular σ is the function ReLU (Rectified Linear Unit): $f(x) = max(x, 0)$.

The reason why deep learning is used here is because it has already been use in multiple other field (like natural language

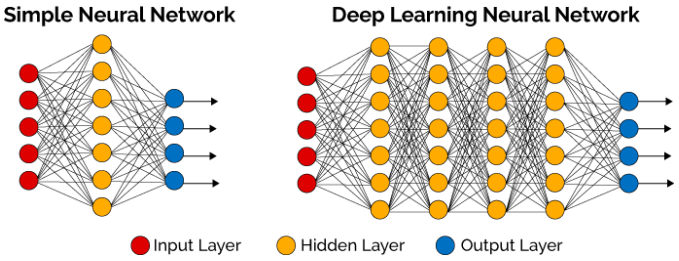


Figure 3: Schema of neural networks. (Credit: `https://towardsdatascience.com/`)

processing, computer vision, audio recognition, speech recognition, etc.) with good results, and this project try to apply it to just another new domain [11, 12, 13, 14, 15].

2.1.2. Recurrent Neural Network

Recurrent Neural Network is a network that, when making a decision, takes into account both the current input and the past inputs. In other terms RNN are networks with memory. RNN needs its input to be a temporal sequence, and it will calculate the output in a causal order. Meaning that, for an input x, if $a < b$ then the output of the RNN of $x[a]$ will be calculated before the one of $x[b]$ and the memory of $x[a]$ will influence $x[b]$ but not the other way around. A Neural Network where no such thing happens is called a feed-forward neural network.

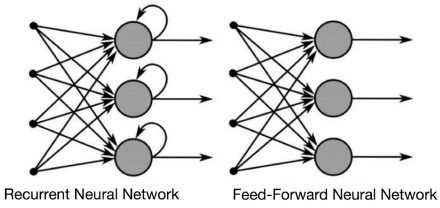


Figure 4: Difference between recurrent and feedforward Neural Network. (Credit: [16])

In the models, a special RNN called LSTM (Long-Short Term Memory) will be used. As its name implies, it is an RNN that can efficiently store information on both short term and long term. It makes it one of the best RNN there is. Here is a little break down of how it works :

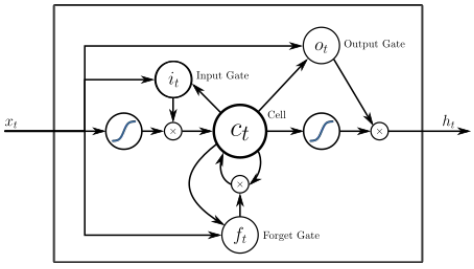


Figure 5: schema of LSTM. (Credit: Wikipedia)

Where c_t is the cell that keeps track of the dependency, i_t is the input gate that controls the extent to which a new value

flows into the cell, f_t is the forget gate that controls the extent to which a value remains in the cell and o_t is the output gate that controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. x are elements wise multiplications and $\int$is the sigmoid function ($f(x) = \frac{e^x}{e^x+1}$).

For the details and equations of both network see [17].

Those methods has some disadvantage, in particular how slow it is, as it can not parallelize operation because the calculation must be done in causal order, and it is hard to adapt the network itself. But the important advantage here, that will be used later, is the ability for the RNN to creates memory vectors, i.e. a vector that represents the past information.

2.1.3. Convolution Neural Network

Above, the concept of fully connected layers, a layer where every elements of the input is linked to every elements of the output, was presented. The Convolution Neural Network (or CNN) is a counter example of such layer. CNN is a model where a specific operation is applied repeatedly over subsets of the input, and can be split into two parts :

The convolution : taken a matrix of size $m \times n$, called filter, that will be fitted by the model :

$$F = \begin{pmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m,1} & f_{m,2} & \cdots & f_{m,n} \end{pmatrix}$$

, and a stride which is the distance between two successive operations, we have :

$$y_{a,b} = \sum (X_{a*sx:a*sx+m;b*sy:b*sy+n} \odot filter) = \sum_{i=0}^{m} \sum_{j=0}^{n} x_{a*sx+i;b*sy+j} * f_{ij} \quad (1)$$

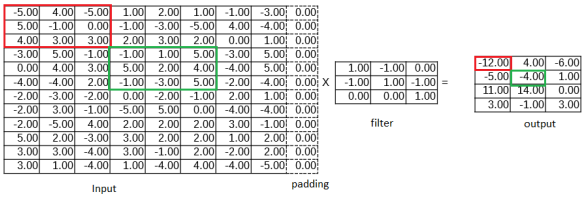
where X is the input, Y is the output, where $y_{a,b}$ is the element of matrix Y on the a^{th} line and b^{th} column, $X_{a_1:a_2,b_1:b_2}$ is the submatrix between $X(a_1, b_1)$ and $X(a2, b2)$:

$$X_{a_1:a_2;b_1:b_2} = \begin{pmatrix} x_{a_1+1,b_1+1} & x_{a_1+1,b_1+2} & \cdots & x_{a_1+1,b_2} \\ x_{a_1+2,b_1+1} & x_{a_1+2,b_1+2} & \cdots & x_{a_1+2,b_2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{a_2,b_1+1} & x_{a_2,b_1+2} & \cdots & x_{a_2,b_2} \end{pmatrix}$$

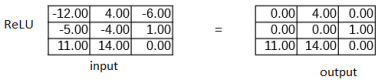
$\odot$ is the element wise multiplication, sx and sy are the strides along the lines and the columns and m and n are the number of lines and columns of the filters.

So there is three hyperparameters in this network: the size of the filter, the strides and the number of filters. If there is more than one filter, the output of the tensor will create a new dimension in which will be stored the different results of the different filters. Sometimes there is not enough place in the input to do the operation because of the size of the filter, in such case a padding is added so that the new dimension of the input match those of the filter.

Here is an example with the filter described, a stride of 3X3 and with padding of 0s used when needed. The red output is the result of the convolution of the red input, idem for the green one.



The second part is the activation function, a function is applied across the matrix so that the model is not linear. The most used function is ReLU : $f(x) = max(0, x)$, and that is the one use for the example below:

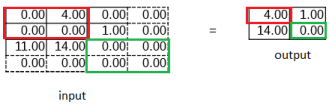


Very often after a convolution network, a pooling layer is added. It is a function that takes a submatrix and extract one number from it. Like the convolution it also has strides between the operation. The hyperparameters of this layer is the strides and the size of the submatrixes. The most used one are max pooling and average pooling, so, with as an example max pooling, the equation of the output is :

$$y_{a,b} = max(X_{a*sx:a*sx+px;b*sy:b*sy+py})$$

where X is the input, Y is the output, max is a function that return the maximum value between the elements of a matrix, sx and sy are the strides along the lines and the columns and px and py are the number of lines and columns of the submatrix to apply the function to.

As an example with pooling size 2X2, strides 2X2 and with padding of 0s used when needed. The red output is the result of the pooling of the red input, idem for the green one.



CNN allows to reduce the input into one easier to process without loosing too much information, and it proved its potential on other problem, the most notable being computer vision, Also, specifically for this problem, CNN does not need to have inputs with a fixed size, which is a good thing because the sizes of songs to analyze are very variable.

2.2. Framework Architecture

The framework used for creating the models is the tf.estimator framework, and is schematized in Figure 6.

The DATA entry is the dataset fed to the model, it then passes through the input function into the estimator that feeds it to the model with one of the three state : TRAIN, EVALUATE and PREDICT.

There are two things that can be customized by the creator of the model :

The input function, input_fn, a method that takes the input and transform it into tensors or datasets, so that it can be fed to the models. It must returns both the features and the labels.

The model function, or model_fn, that takes the features, the labels, the hyperparameters and the current state, and that returns the element asked from the state.

If the state is TRAIN, the model must return an optimizer from the loss of the model. If it is EVALUATE, the model must return the loss and some evaluation measures deemed pertinent for evaluating the model. The evaluating measures returned depends on the type of loss chosen, see section 3.3 for the details. Those two states can be run alternatively to train the model while keeping an eye on how the learning is going.

If the state is PREDICT, the model can not take the labels, and must predict one of its specific element. Here, because of how similarity, loss and metrics are defined, the prediction to return is the embedding of the music, i.e. vectors representing the musics such that similar musics are equivalent to similar embeddings.

The interest of this framework is : it allows a clean code, especially for the train/evaluation part, it also allows to have graphical representation of the training with tensorboard, it includes the early stopping (method that consist in stopping the training when it stops to improve the accuracy of the model) and it allows us to do the training, testing and evaluating on the same model.

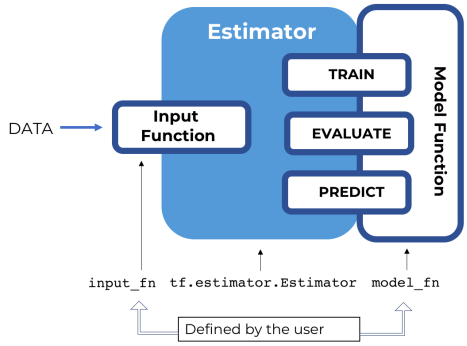


Figure 6: Schema of tf.estimator. (Source: towardsdatascience.com)

3. Model and Problem Statement

3.1. Music Similarity

3.1.1. Problems

The first part to take into account for this project is the question of: what are two similar musics? Indeed, if several peoples are asked what they consider to be similar musics, their answer will not be unanimous. Some might put more importance on the lyrics, some on the melody or the rhythm, some will go to a more personal definition saying that two songs alike are two songs they like or two songs they dislike, and if we go to more artistic people we could get more abstract answer like the feeling of a song. That question is not as much a philosophical problem to be given to the humanities to ponder and more of the question of what will be used as a dataset to create the models. This section will be dedicated to giving the answers used for this project and those rejected. Section 6.2.2 proposes further datasets that could be used in the future.

Problem 1 (Definition of Similarity)**.** *Similarity does not have a mathematical definition and can have multiple interpretation.*

Incidentally, that difference of similarity raises the question of what is a good similar model : does it have to be able to be trained on those multiple definition or do we have to choose one and tailor the model to it ? In this project the general approach was used, because there is not yet a method, available to the lab, to determine what is a good recommendation system. A way to test them would be either to ask people which is better (with survey or with the help of an audio streaming) or to ask some social science labs what would be the best options.

Problem 2 (Multiple Similarity)**.** *Because Similarity does not have one definition, the model should be able to be fit on multiple labels.*

3.1.2. Solutions Used

First approach. The first approach to determine the similarity between the music is the genre, i.e. if a music is rock or if it is metal ... With this end goal, a music is considered similar to another if they share the same genre. The big advantage of this approach is that there is plenty of datasets used to train the models. The drawbacks are that it is very similar to just creating genre classification embeddings and that it is a simplistic task that would normally not require such complex methods to achieve. It is why this method is only used to test the models once they are finished, to see how they fare on this simple comparison, but not to train them.

It must be noted that even if we use genre for this approach, classical genre classification methods can not be used because the model that is being created must be generalizable to other similarity frameworks whose labels are continuous or have too many categories to be clustered.

Second approach. The second approach is to compare the metadata of the music like the author or the album it is in. The big advantage of this method is that it can be done on most datasets of music. The problem is that to train some sort of recommendation system with such datasets would only give similar authors, which is not an improvement on the old ones.

Third approach. The third approach is to have details on the music and use how many details two musics have in common as a similarity metrics. Such details could be the genre of music, the number of singers, the instruments. An example for the details that can be found is the dataset magnatune detailed in section 5.1.1. This method gives a quite general approach of the definition of similarity and that is why, even if the multitude of possible differences makes it a little complex, it is the main definition used for the training.

But there is a question : how precise is this metric ? Is a jazz song with one singer as close as a rock song with one singer

than it is of a jazz song with two singers ? It raises the question of the weight of the features. So to get around this problem, while using this technique, the similarity between two songs become binary with a similarity of 1 if enough characteristic of the songs are in common and 0 otherwise.

3.1.3. Recommendation Systems

With all the talk about recommendation system, it might be thought that it could be used as labels, but that method has a considerable disadvantage : it is too biased. If a model were to be trained on those labels, the principles of machine learning would make that the aforementioned model would tend to consider similar the musics advised by the recommendation system. Meaning that it would have the flaws present in the original system, like the difference of recommendation between the mainstream and niche musics, which was what this approach was supposed to avoid. (For more details on biased datasets : [18])

Also, while the recommendation systems are talked about, it is a good moment to talk why there can be no direct comparison between them and the similarity approach : they can not be compared on a recommendation framework because there is no model that could allow to test the best system (if they were, it could just be used as labels), and they can not be compared on a similarity framework because recommendations system return musics that would be liked, not similar ones and that would put it at an unfair disadvantage.

3.2. Music Comparison

Now that we saw how we can define the similarity between two musics, we should look at the challenges created by comparing musics and so what a model should overcome to be a good model.

The first and most important issue that needs to be solved is the one of the size. With a default frequency of around 44KHz and most musics being around 3 minutes long, the order of magnitude of the number of float needed to represent a music is in the millions. And this causes one big problem :

To understand it, we first need to understand one concept, the curse of dimensionality [19], that can be resumed as "after reaching a certain point, the more dimensions there is in the features space, the more the distances between 2 points becomes effectively random". For the algorithm it means that the bigger the vectors to compare at the end of the model is, the more certain it is that it will not train effectively. With effects beginning when the compared vectors' size are in the hundreds, this mean that the neural network needs to substantially reduce the number of elements during its run.

Problem 3 (High reduction)**.** *The size of the embeddings need to be much smaller than the size of the input.*

The second one is that, counter to other inputs like pictures or texts, the music's representation has only one dimension. It is a problem for different reasons : for one, if left like this there is a lot of classical methods that are not pertinent to use, like RNN for reason that will be developed in Section 4.2. The second is that it is not possible to just reduce the music with the model because if it finishes with only one element it would not be able to represent the complexity of the similarity. This means that somewhere in the model, a dimension need to be added with a big enough size.

There are two ways to do it, the first one is to add size in the preprocessing, like in the algorithm of Section 4.2, or in the model itself, like in the algorithm of Section 4.3.

Problem 4 (Dimension Need)**.** *The embeddings need to have enough dimensions to represent the complexity of the musics.*

Finally, the last challenge stems from the fact that the music does not all have the same size. For example, in the FilmMusic dataset, the minimum length is 10 seconds and the maximum length is more than 3 minutes (and some other might have music even longer). This means, the algorithm should be able to analyze musics with a factor 20 between the size of the smallest and the biggest.

Problem 5 (Variable Dimension)**.** *The model should be able to analyze songs of very different sizes.*

In summary, a good model needs at least to be able to reduce the size of the input but not too much while not depending too much on the size of the input.

3.3. Learning to Rank

The aim of the model is to takes into input a music, a dataset and return one, or more, similar musics. That task is similar to a search engine using Machine Learning (even if, usually, the query is smaller than the text searched). There is already an existing field for such task and it is called Learning to Rank [20], whose subject is to find the best loss function for such model. We will see in this section an introduction to this subject.

There is a case to be made that this project should be only on similarity and that there is no need to complexify it further, so this part will also show how a good Learning to Rank model will also create a better similarity model.

In Learning to Rank, there are three types of approach for the loss functions :

- The pointwise approach where the loss only look to the queried music and one compared music for each iteration. This strategy consist in reducing the problem into a regression or classification.
- The pairwise approach where the loss look to the queried music and two compared musics for each iteration and try to determine which one is the most similar to the query.
- The Listwise approach where the loss look to the queried music and a list of compared musics (more than two) for each iteration and try to find the best permutation over the list, i.e. which is the closest to a list sorted by the true similarity between the songs and the query.

Between those three approaches, one is particularly worst than the others and that is the pointwise one. One way to see it is to look at the wikipedia article which has algorithms wrote in 2019 but whose last purely pointwise method dates from 2007. There is no paper that clearly explains why that is, but it can be argued that either pointwise labels are not precise enough for the task or that comparing elements between each other allows much stronger learning. So this method will not be taken into account when making the models.

Between the two other approaches, listwise tends to give better result. The reason might be that a lot of pairwise algorithm use not the distance of similarity extracting from the labels but only which one is most similar (maybe because, using the exact value of the labels might create more bias from the dataset than just comparing them two by two). In this disposition, listwise methods would penalize errors that a pairwise ones would not find.

Example 1. As an example let's take documents 1, 2, 3 and 4 representing their rank in a sorted similarity list. Between the two result of a search : 3124 and 1342, 1342 is the best because the first answer is the most similar and as a list it can be recognized. But as pairs, there is the same number of false pair in both search (3-1 and 3-2 in the first and 3-2 and 4-2 in the second) and it would not be differentiated pairwise even when one is better than the other.

But the pairwise has still some advantages on some situation. If it takes musics three by three (query and two musics compared) more complex method and bigger music can be trained and tested compared to the size needed for the numbers of music needed with the listwise approach. That is why an option to use pairwise learning will be available on the final model. Also, as a metric, the pairwise comparison is still an interesting one to have.

Problem 6 (Ranking Algorithm)**.** *The model created should be be equivalent to either:*

- *a model that, given a song to compared and a list of songs, should be able to order the list by similarity to the compared song. (Listwise)*
- *a model that given a song to compared and a pair of songs, should be able to determine the best one. (Pairwise)*

So the function used as a loss for the models are:

- For the pairwise, with as input the query q, the musics m^+ and m^- with m^+ being more similar to the query than m^-, and d a function that calculate the distance between two musics, the loss is:

$$max(0, 1 - d(q, m^-) + d(q, m^+)) \tag{2}$$

- For the listwise, lambdaloss [21] is used, it is an algorithm created by google with the aim of creating a loss that, the closer it gets to 0, the closer the NGCD (one of the best listwise metric, see the section 5.1.2) gets to 1 (the maximum of the NGCD).

It can be noted that not a lot of research was made on those losses, the reason is that states of the art algorithms were used and that the main subject of this project is more about similarity than learning to rank.

4. The Algorithm

This part will be about the algorithms used for the models. The aim of the algorithms is to have as input musics (raw or preprocessed) and as output embeddings of those musics. Once they are created, they can be compared between each other (here using the euclidean similarity or the cosine similarity to prevents problems from the norms of the vectors) to be fed, to the losses to be trained or to the metrics to be evaluated and tested. This way allows, once the model is done, to do the embeddings in preprocessing which makes for faster searches.

4.1. Librosa Model

The aim of this method is to create a method that do not use deep learning but that is accurate enough to be compared with models that use it. This method is called librosa, from the library used to extract all the music information retrieval (MIR) features.

The first part is to extract the data, from each music is extracted : the zero-crossing rate of the audio time series, the Constant-Q chromagram, a chroma variant "Chroma Energy Normalized" (CENS) [22], the tonal centroid features (tonnetz) [23], the chromagram from the absolute values of the Short-time Fourier transform (STFT), the Root Mean Square Error (RMSE) of the STFT, the spectral centroid of the STFT, the p'th-order spectral bandwidth of the STFT, the spectral contrast [24] of the STFT with 6 frequency bands, the roll-off frequency of the STFT, the mel-scaled spectrogram, the Mel-frequency cepstral coefficients.

For each of those features, over each dimensions were calculated the mean, median, minimum, maximum, standard deviation, skewness (a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean) and kurtosis (fourth central moment divided by the square of the variance).

At this point, a music was reduced to a simple vectors, but it has hundreds of dimensions and because of the curse of dimensionnality [19] (see section 3.2), we can not compare music directly with it. First, the number of dimension of must be reduced.

For that, a Principal component analysis (PCA) [25] is applied on the dataset of embeddings. This algorithm consists in standardizing the dataset, creating its covariance matrix and calculating the eigenvectors and eigenvalues of this covariance. For each eigenvector, its eigenvalue can be seen as representing how important it is for approximating the dataset. So taking the eigenvectors whose eigenvalues represent more than 99% of the total of the eigenvalues will create a representation precise and little enough so it can be compared. In the case of this project's dataset, the embedding's size can go down to less than 10 values.

As written above, this method is used as a non machine learning method. So it is also used while creating the dataset to add musics whose similarity can not be determined by such direct process in the aim of preventing the model from being only a more complex version of this algorithm, and to prove that those are not just dataset errors and that deep learning can better extract some information.

4.2. Unsupervised Recurrent Neural Network

In the first time of this project, datasets were a little lacking and so unsupervised training was tried. This approach was abandoned once bigger datasets were found because of its result and because deep models were more promising. Nonetheless, it allows the creation of general embeddings of music that could be used in other models, by example as input.

The input of the model is a chromagram extracted from the music on which was applied a Short-time Fourier transform. The idea of the chromagram is to try to extract the 12 pitches (C, C#, D, D#, E ,F, F#, G, G#, A, A#, B) at every moment. This is the closest thing to pitch extraction and that is why it was used. This also means that the size of the music after the preprocessing is ? X 12, where the ? is the length of the music after being processed.

Once the preprocessing is done, a model is trained that takes a music and try to predict the next STFT vectors knowing all the ones that came before using a LSTM (see section 2.1.2 for details). The loss used is the sum of the absolute difference between an element and its prediction.

So even if the learning itself is not unsupervised, the unsupervised here come from the fact that to extract the similarities, there is no need to already have similarities between the musics.

There is two hyperparameters in this model : the learning rate and the dropout, which is few and that allows to easily get the best parameters by just testing a wide range.

Once this model is trained enough, music is fed to the model and the states and results are extracted to create the embedding. The idea is to consider the states used to predict the next elements in a LSTM, as able to extract information from the music.

Because the LSTM needs to know the length of the sequence on which it is training itself, the input was split into sections on 1000 elements. Padding is added if such splits can not be implemented directly.

Once the embedding are extracted, there are still a way to optimize the model : first, because the music was split into sections, there is, at the end, results for multiple part of the songs, so there is a need to reduce them to just one vector and that creates the first optimization : to find the best function between the mean, the maximum and the last function (only takes last result). After reducing the result, it must be determined which parts to keep between the result and the states.

4.3. Wavenet

This algorithm adapts the new algorithm wavenet [26] that is used to predict a song float by float in its mp3 format. When the project was done there was no official code with the paper nor were they description of the hyperparameters used (notably some activation functions).

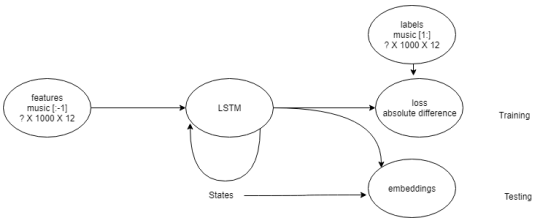


Figure 7: Scehma of the rnn model

Design choices. Before presenting the details, there are two variations on Convolutional Neural Network that needs to be discussed :

- Causal neural network in which the results obtained by a CNN is only influenced by elements that came before. Concretely, it means that the padding will be put at the beginning and not at the end, as it is usually done. It was used because of the prediction, so that the next element is only predicted from the element before it. But for a similarity model, this is not a big difference to classical CNN. Here is a representation of stacked Causal CNN. See Figure 8.

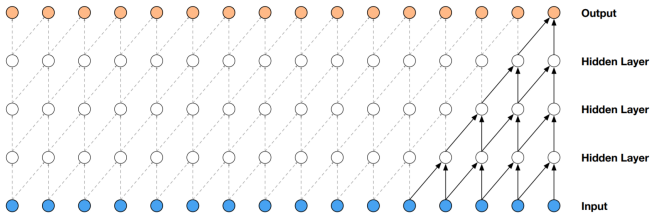


Figure 8: Visualization of a stack of causal convolutional layers. (Credit [26])

- A dilated convolution is a convolution in which the operation is applied over an area larger than the filter by skipping input values. Here is a representation of stacked dilated causal CNN (dilation of 1 is a classical CNN). See Figure 9.

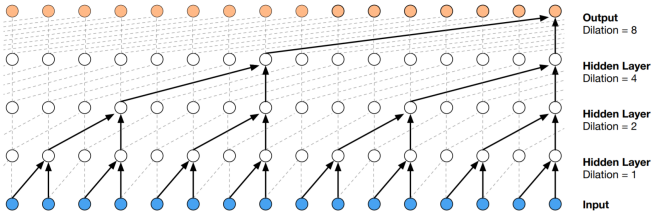


Figure 9: Visualization of a stack of dilated causal convolutional layers. (Credit [26])

Original Wavenet. Wavenet takes as input the raw decompressed music from the mp3 files and is based on this structure of the dilated causal neural network. It makes use of the fact that with every layer the number of elements that are linked to the output grows exponentially. Wavenet base is to use those

CNN and apply as an activation function :

$$f_k(x) = tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x)$$

Where $W_{*,k}$ are the filters of the convolution of the k^{th} layer, σ is the sigmoid function, tanh is the hyperbolic tangent function, $*$ denotes a convolution operator and $\odot$ denotes an element-wise multiplication operator. Once a layer is done the results will create two versions : the one that will be part of the results (called skip results) and the one that will be the input of the next layers. Once all the convolutions are done, it outputs the skip results of each layer which is then pooled so that it goes back to a one dimension vector and finally process again so that it gives the next byte for the song.

A representation of the model can be found in Figure 10. Where the + element represents an element-wise addition, the x element represents an element-wise multiplication, σ represents the sigmoid function, tanh the hyperbolic tangent funtion, ReLU a rectified linear unit layer, and the rectangles with numbers inside represent a convolution neural network with the size of its filter, by example the rectangle between the element-wise addition and multiplication represents a CNN with a filter of one column and one row.

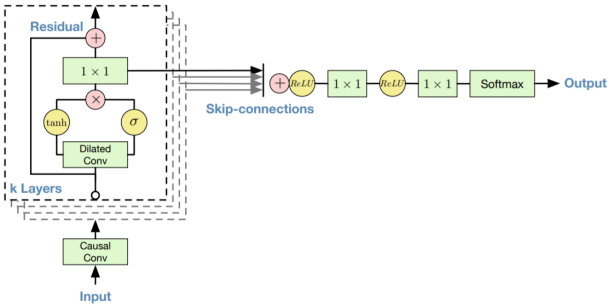


Figure 10: Overview of wavenet. (Credit [26])

Our Wavenet-based solution. For the use of the similarity, the loss function is not needed, also summing the results of the convolution is only useful when trying to reduce the dimension over the song to one, but, for the similarity, creating a matrix out of the different results seems more interesting. To simplify : only the idea of cascading dilated causal CNN is kept from the model. The unit of the dilated convolution with the activation function will be called a wavenet layer (Figure 11).

From the similarity also comes the need to reduce the number of elements, there are two moments where it is pertinent to do so :

- *Between the layers:* we consider the stacks of dilated CNN as one complex CNN, and we apply a pooling function between them. For taking full advantage of the structure of the model, there should not be a pooling function after each layers of wavenet but at regular interval. Because the skip connections will be concatenated at the end, the pooling function is applied on all of them even if there has been no change since its last application. A series of wavenet layers followed by a pooling function will be called a wavenet stack and is schematized in Figure 12:

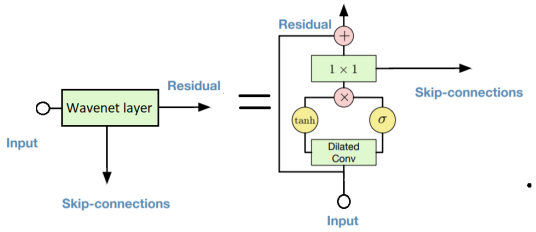


Figure 11: Schema of a wavenet layer.

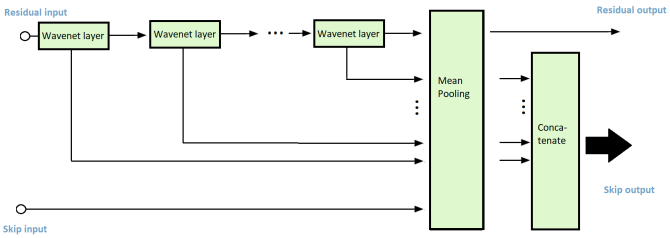


Figure 12: Schema of a wavenet stack.

- *At the end of the stacked layers*, once it was already reduced, a last reduction is applied in the form of a classical CNN and pooling so that, whatever its original size was, the song is reduced to an embedding whose size is fixed and considered fitting for the task. Some full layers can then be added to better fit the model. This model is schematized in Figure 13.

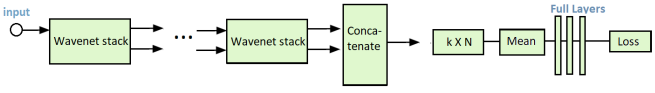


Figure 13: Schema of the wavenet similarity model.

Where k is the number of wavenet layers used in the wavenet stacks, N is an arbitrary value and loss is one of the loss used by the model as defined in Section 3.3.

- *Pre-processing:* As said before, the only preprocessing used for this model is the one consisting into decompressing the mp3 files into a format that can be fed to the model. But this model is so big that it often causes memory error, especially when using the GPU. So a preprocessing method that was used was to apply a mean pooling of size 5, and try to mitigate the loss of precision by creating more complex models.

- *Hyperparameters:* So from the models, the hyperparameters are: the learning rate, the dropout, the pooling function used, the number of layers and their dilations, where

to apply the reduction between the layers and the reduction used at the end. Also, the size of the filter and evolution of dilation by layer can be changed. By example instead of a filter of size 2 and multiplying the next layer's dilation by 2, the filter size becomes 4 and the next layer's dilation is multiplied by 4, which keep the idea while reducing the number of layer.

5. Experiments

5.1. Experimental Settings

5.1.1. Dataset

Three datasets were used :

- The FilmMusic dataset, that was given by the supervisor. It is a random sample of music from : https://epicmusicvn.sourceaudio.com/ . It contains 1727 songs. The similarity used was the if the songs belong to the same album.
- The zalo_music_genre dataset that comes from : https://challenge.zalo.ai/portal/music . It contains 5158 songs. The similarity used here is if the songs has the same genre. There are 10 genres given in the dataset.
- The MagnaTagATune dataset that comes from : http://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset . It contains 25,860 songs. In the dataset, details were given, in annotations_final.csv, about the song, such as the instrument used, the number of singer. This was used as similarity.

Details of the similarities are discussed in Section 3.1.2.

The best models from each algorithm were trained and evaluated only on one dataset before being tested. The reason for that is that, if we estimated the best hyperparameters on the three dataset, it would take thrice the time every time we tried a little modulation. It is much faster and more efficient to evaluate the hyperparameter on only one dataset and to test after if the best model obtained is generalizable to the others.

The dataset used for that is MagnaTagATune because : it has more elements that the two other, so it has fewer chances of overfitting, the songs are generally smaller so every step is quicker and because the similarity is complex, the results of this model show more if the model can extract information from the songs.

5.1.2. Evaluation Measures

Pairwise comparison, or PC. takes one song, one similar to the first and one which is not and test if the models find which is which.

The Normalized Discounted Cumulative Gain. at 1, or NDCG@1, and the NDCG@6 (6 being the length of the classical list by query). Those are one of the most classical metrics to evaluate Learning to Rank models.

Given list of true relevance sorted by the relevance calculated by the model :

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

where rel_i is the relevance of the i^th element of the list. And the NDCG that will be used as a metric is :

$$NDCG@k = \frac{DCG@K}{IDCG@K}$$

where IDCG@K is the ideal DCG, i.e. the maximum value that can be obtained with a permutation of the list, meaning the DCG of the list sorted by their true relevance.

The metric penalizes the permutations that do not order the predictions on the true sorted relevance. The results are between 0 and 1 where 1 is the best result.

The @K is the number of sorted elements that should be taken into account. Meaning that NDCG@1 test only the first element whereas NDCG@6 tests all the list.

The mean average precision. or MAP, which takes into input a list of binary similarity (elements that are 1 if they are similar and 0 otherwise) sorted by the prediction of the models, and it calculates the mean over the dataset of the average of the precision of the first n element.

With

$$P(k) = \frac{\sum_{i=1}^{k} y_i}{k}$$

where y_i is the relevance at position i. And the average precision is :

$$AP = \frac{\sum_{i=1}^{n} P(i) y_i}{\sum_{i=1}^{n} y_i}$$

The metric MAP is the mean of the AP over the whole dataset.

Because this metric asks for binary similarity, dataset that are not binary can not be tested directly and need to set a threshold that will determine if a song is similar or not. This might cause some imprecision.

5.2. Results

Here are the results of the different models. The random model represent a model where ranking is given randomly and the worst model is a model where the ranking is the worst ranking possible. They are here to compare the created models with bad ones.

Ranking. To test the models, for the ranking metrics : for each song, we randomly take three similar music and three non similar ones, we compare them to the original song, and we score the ranking they produce with the ranking metrics (see Section 5.1.2). For the pair metrics : for each song, we randomly take fifteen pairs of similar and non similar musics, compare the pairs to the original music and apply the pairwise comparison.

When we look at the result by dataset, we can see that as a whole the results are the best in FilmMusic, with results in MagnaTagATune being not far behind, and the worst results are

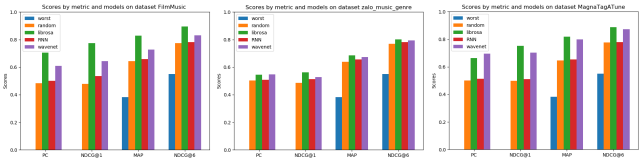


Figure 14: Score of the models

given in the zalo_music_genre dataset. This shows the complexity of every similarity definition chosen, with album similarity detection being the easiest and genre similarity detection being seemingly the hardest to determine. It is unknown exactly why the genre dataset ranks so poorly. It might be because : the complexity of genre detection was underestimated, the genre might not be a good similarity definition or the musics of the dataset are skewed in a way that prevent good similarity comparisons.

As expected, librosa, with the use of all the best MIR methods, gives very good result on most dataset, and it proves its use as a good method to compare other method with.

The RNN model's result are bad. Most of the time it is as bad as random, and when it is not, it is not much better. It is one of the reason that this model was dropped, maybe one reason it is so bad is that the LSTM takes information on a too little length to represent the model. But even with that, the model still has potential, section 6.2.2 proposes way to improve the results from the embedding of the model or of the model itself.

Comparison with librosa. When we look to the wavenet results compared to the librosa model, the first thing we see is the impact of the dataset size on the deep learning model : in the FilmMusic results(on which the training is done on less than one thousand songs), librosa have a clear difference compared to wavenet that is not as present in the MagnaTagATune results(on which the training is done on around ten thousands songs). So the wavenet model should be judged on the MagnaTagATune dataset, where it at its best.

If we judge it that way, it is close to be as good as the librosa model, except on pairwise comparison where it has a better score. Even when the model is trained on smaller dataset, the results are comparatively better than the rnn model and the random model.

We can also notice the difference of results between the NDCG@6 and NDCG@1, where the NDCG@1 of the wavenet model is always comparatively less precise than the NDCG@6. The reason for this result is that wavenet was trained to have the best ranking over all its list and so locally, the precision of the first element is not optimized. It is a weakness of the loss function used, even if, in the long run, if the NDCG@6 keeps going up, the NDCG@1 would also go up.

Finally, the results of MagnaTagATune shows us one thing : the difference between the rankwise and pairwise metrics. Indeed, even if librosa has better scores on every ranking metrics, the wavenet is the best when it comes to pair comparisons.

Skew issues. Next, let's see what happen when we test the machine learning algorithms on a dataset that is very skew against the librosa approach : we create a dataset that, for the rankwise metrics : for each song, get the three similar songs with the worst score and the three non similar song with the best ones. For the pairwise ranking : creates fifteen pairs where the n^{th} pair contains the similar songs with the n^{th} worst score and non similar song with the n^{th} best one. Here are the results :

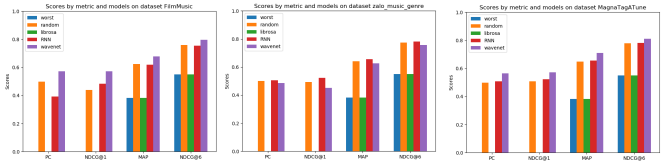


Figure 15: Score of the models on failed librosa model.

First those results confirm the complexity of each similarity definition by having the same ranking of results than the original ones.

But we also see that every created model becomes worse on this dataset. That means that those comparisons and ranking are harder to do, but we also saw that, in most parts, the wavenet model is still able to have more than average results.

Wavenet Variations. At the end of the project new results (Figure 16) were obtained from variations on the wavenet model and allowed to get better scores overall. The main variation was the use of euclidean similarity instead of cosine similarity to compare the embeddings. The definition of the euclidean similarity between A and B is defined as :

$$d(A, B) = -\|A - B\|$$

where $\| * \|$ is the euclidean norm.

An other improvement was to use the fully connected layers at the end of the model to reduce the size given by the last pooling. But because it was made to late, it was only trained on the MagnaTagATune and so the only score available are on this dataset. The old wavenet model and the librosa model were added for comparison.

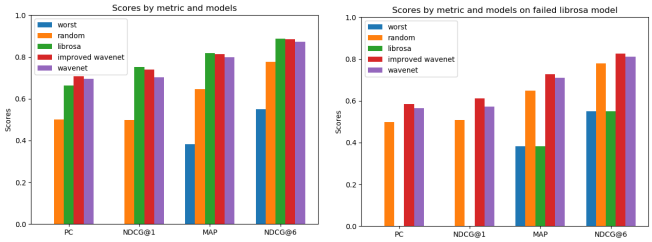


Figure 16: Score of the improved models.

6. Conclusions

6.1. Summary of the Work

In this project, we presented the problem of similarity, and proposed a framework and different models that tried to solve it. In experience, we saw how good the models were doing with

one, wavenet, which managed to be almost as good as the referential model. It shows the potential of the use of deep learning as well as the usefulness of adapting prediction models for the creation of similarity algorithm.

But even when showing the potentials of those methods, one can ask why the model did not show better results, especially since one used wavenet, a state-of-the-art algorithm. As always in machine learning, the reason are not certain, but here are some guess :

- This model is not yet completely optimized : until the very end, better and better results were obtained, it is likely that given more time the results would keep going up.
- The model was created from a music generating model, this means that it was not optimized for our task and it might create imprecision in the results.
- Maybe the definitions of similarity that we used do not always allow good comparisons from only the content of the song. It is evident in the Zalo dataset, but it might also be the case, to a lesser extent, in the other ones.
- The limitation of the equipment used for the training might be at fault. To be able to fit a large number of elements within a reasonable timeframe, the GPU must be used. But a GPU is not originally used for such task and it does not have a lot of memory available for it. This is especially troublesome because of the size of a music and the fact that several are needed in the loss function. An example of this limitation is the use of the mean pooling at the beginning of the wavenent model to be able to have more complex algorithm.

If those reasons are correct, it means that further improvement are possible. The next section is dedicated to suggestions of amelioration for future work.

6.2. Discussions and Future Work

6.2.1. Datasets

Because of time and equipment of this project not every method was used. This part are propositions of dataset that can be studied or used in future project.

We have just seen that some dataset can not be used because they are biased. The use of recommendation system was raised but, in a broader way, most of the labels that can be collected from the logs of a music provider is biased because what the users listen to is also filtered by the recommendations, but that does not mean they are unusable. It seems that methods are being developed to train models on biased dataset like here [27]. So, if the lab manage to be sponsored by a platform of music that allow to access its log, project could be made to try to debias it.

One method of dataset creation that was attempted (but drop because I did not have enough spaces left on my computer) was to use spotify's playlists to extract similarities. Counter to most data extracted from music platforms, the playlists are not bias, or not so much that it would prevent their use in machine learning, because even if the recommendation systems influenced the songs used, creating a playlist consist in grouping together music that are somewhat similar. Moreover, if, using the search API of spotify, we search for playlists whose name would not imply a vague link between the songs (like "favorite").

Once the playlists are obtained, there is a need for a way to determine how similar musics are from one another. This could be another project, but while building the dataset the method used was Word2vec [28]. Word2Vec is originally a machine learning algorithm used to create embeddings on words given a corpus of documents. Its learning is based purely on giving similar embeddings to words often close to each other and opposite embeddings to word that are often not. If we consider the musics as words, the playlists as document and apply this algorithm, we obtain a model that gives similar embeddings to songs that are frequently in the same playlists and opposite to songs that are not.

This model has two hyperparameters : the size of the embedding and the maximum number of word that can be considered close. In the dataset created, the size is 300, because it is a common size for word embedding, and the maximum is 100, after that it is considered that music are too far away to be related with enough precision. If this dataset is to be used, those hyperparameters should be tuned.

Legally, the Terms of Service are here : [29]. I did not find any term that would prevent the use of the api in a non-lucrative way for public research focusing only in the content of the playlists (i.e. not about the users).

6.2.2. Algorithm

To improve the RNN model, adding layers might be tried, but it is not sure that it would work. If training is done on more than the LSTM, and even if the precision of the model to predict the next vector goes up, the precision of the LSTM might locally go down, and so would the precision of the extracted information.

Instead, to improve this method, other deep learning method that have memory can be tested (like the Attention networks).

One way to ameliorate the results of this algorithm is to use other input : either takes other MIR method for preprocessing, or creates input using other unsupervised learning, by example an auto encoder.

To improve the wavenet model the first thing to do would be either to delete the mean pooling use by the preprocessing or to find better hyperparameters for every part that is not yet parameterizable in the tensorflow model. Trying other pooling at the middle or at the end might also be useful. It can also be tried to use the dilated CNN from the wavenet layers to reduce the size of the songs, but it would mean rethinking how the stacked layers might be combined.

An other thing to try would be to train the wavenet as it was initially planned (for prediction), and once it we have a good model, take only the stacked layers and try to train the model with pretrained layers.

Finally, as shown in the discussion of the results, the model get better when it is trained on our biggest dataset. Maybe train-

ing the model on even bigger dataset would give even better results.

Acknowledgement

I would like to express my deep and sincere gratitude to my research supervisors, Prof. Karl Aberer and Thanh Tam Nguyen, for giving me the opportunity to do this project and for the help given throughout.

References

[1] M. Madathil, Collaborative filtering.
URL http://hpac.rwth-aachen.de/teaching/sem-mus-17/Reports/Madathil.pdf

[2] A. van den Oord, S. Dieleman, B. Schrauwen, Deep content-based music recommendation, in: C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26, Curran Associates, Inc., 2013, pp. 2643–2651.
URL http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf

[3] W. Wang, H. Yin, Z. Huang, Q. Wang, X. Du, Q. V. H. Nguyen, Streaming ranking based recommender systems, in: SIGIR, 2018, pp. 525–534.

[4] N. Q. V. Hung, D. C. Thang, N. T. Tam, M. Weidlich, K. Aberer, H. Yin, X. Zhou, Answer validation for generic crowdsourcing tasks with minimal efforts, VLDB J. (2017) 855–880.

[5] N. Q. V. Hung, H. H. Viet, N. T. Tam, M. Weidlich, H. Yin, X. Zhou, Computing crowd consensus with partial agreement, TKDE (2018) 1–14.

[6] N. Q. V. Hung, H. Jeung, K. Aberer, An evaluation of model-based approaches to sensor data compression, TKDE (2013) 2434–2447.

[7] H. Yin, L. Chen, W. Wang, X. Du, N. Q. V. Hung, X. Zhou, Mobi-sage: A sparse additive generative model for mobile app recommendation, in: ICDE, 2017, pp. 75–78.

[8] N. T. Tam, M. Weidlich, D. C. Thang, H. Yin, N. Q. V. Hung, Retaining data from streams of social platforms with minimal regret, in: IJCAI, 2017, pp. 2850–2856.

[9] H. Yin, N. Q. V. Hung, Z. Huang, X. Zhou, Joint event-partner recommendation in event-based social networks, in: ICDE, 2018, pp. 1–12.

[10] H. Yin, H. Chen, X. Sun, H. Wang, Y. Wang, Q. V. H. Nguyen, SPTF: A scalable probabilistic tensor factorization model for semantic-aware behavior prediction, in: ICDM, 2017, pp. 585–594.

[11] N. Q. V. Hung, D. C. Thang, M. Weidlich, K. Aberer, Minimizing efforts in validating crowd answers, in: SIGMOD, 2015, pp. 999–1014.

[12] H. Yin, Z. Hu, X. Zhou, H. Wang, K. Zheng, N. Q. V. Hung, S. W. Sadiq, Discovering interpretable geo-social communities for user behavior prediction, in: ICDE, 2016, pp. 942–953.

[13] H. Yin, X. Zhou, B. Cui, H. Wang, K. Zheng, N. Q. V. Hung, Adapting to user interest drift for POI recommendation, TKDE (2016) 2566–2581.

[14] N. Q. V. Hung, N. T. Tam, N. T. Lam, K. Aberer, An evaluation of aggregation techniques in crowdsourcing, in: WISE, 2013, pp. 1–15.

[15] N. Q. V. Hung, K. Zheng, M. Weidlich, B. Zheng, H. Yin, N. T. Tam, B. Stantic, What-if analysis with conflicting goals: Recommending data ranges for exploration, in: ICDE, 2018, pp. 1–12.

[16] W. D. Mulder, S. Bethard, M.-F. Moens, A survey on the application of recurrent neural networks to statistical language modeling, Computer Speech and Language 30 (1) (2015) 61 – 98. doi:https://doi.org/10.1016/j.csl.2014.09.005.
URL http://www.sciencedirect.com/science/article/pii/S088523081400093X

[17] A. Sherstinsky, Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network, CoRR abs/1808.03314. arXiv:1808.03314.
URL http://arxiv.org/abs/1808.03314

[18] A. Woodie, Three ways biased data can ruin your ml models, Datanami.
URL https://www.datanami.com/2018/07/18/three-ways-biased-data-can-ruin-your-ml-models/

[19] B. Shetty, Curse of dimensionality, Towards Data Science.
URL https://towardsdatascience.com/curse-of-dimensionality-2092410f3d27

[20] H. Li, A short introduction to learning to rank, IEICE TRANS. INF. SYST. VOL.E94–D, NO.10.
URL http://times.cs.uiuc.edu/course/598f14/l2r.pdf

[21] X. Wang, C. Li, N. Golbandi, M. Bendersky, M. Najork, The lambdaloss framework for ranking metric optimization, in: Proceedings of The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), 2018, pp. 1313–1322.
URL https://storage.googleapis.com/pub-tools-public-publication-data/pdf/1e34e05e5e4bf2d12f41eb9ff29ac3da9fdb4de3.pdf

[22] S. E. Meinard Muller, Chroma toolbox: Matlab implementations for extracting variants of chroma-based audio features, 2011.
URL http://resources.mpi-inf.mpg.de/MIR/tempogramtoolbox/2011_MuellerEwert_ChromaToolbox_ISMIR.pdf

[23] C. Harte, M. Sandler, M. Gasser, Detecting harmonic change in musical audio, in: Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia, AMCMM '06, ACM, New York, NY, USA, 2006, pp. 21–26. doi:10.1145/1178723.1178727.
URL http://doi.acm.org/10.1145/1178723.1178727

[24] L. L. H.-J. Z. J.-H. T. Jiang, Dan-Ning, L.-H. Cai., Music type classification by spectral contrast feature, 2002.
URL https://hcsi.cs.tsinghua.edu.cn/Paper/Paper02/200218.pdf

[25] Z. Jaadi, A step by step explanation of principal component analysis, 2019.
URL https://towardsdatascience.com/a-step-by-step-explanation-of-principal-component-analysis-b836fb9c97e2

[26] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, K. Kavukcuoglu, Wavenet: A generative model for raw audio, CoRR abs/1609.03499. arXiv:1609.03499.
URL http://arxiv.org/abs/1609.03499

[27] H. Jiang, O. Nachum, Identifying and correcting label bias in machine learning, CoRR abs/1901.04966. arXiv:1901.04966.
URL http://arxiv.org/abs/1901.04966

[28] B. Shetty, Introduction to word embedding and word2vec, Towards Data Science.
URL https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

[29] Spotify, Spotify developer terms of service, Spotify.
URL https://developer.spotify.com/terms/