



UNIVERSITAT DE
BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFOMRÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

WORD2VEC EMBEDDINGS
FOR PLAYLIST
RECOMMENDATION

Autor: Anna Bach

Directors: Dr. Santi Seguí i Guillem Pasqual

Realitzat a: Departament de Matemàtiques i
Enginyeria Informàtica

Barcelona, June 2018

Abstract

We present an ML approach to musical playlist recommendation. Using the algorithm Word2Vec, a shallow two-layer neural network trained to reconstruct linguistic context of words, we have created several embeddings using tracks and playlist titles as words of an artificial vocabulary. Some experiments with different trade-offs between the diversity and the popularity of songs in playlists are analyzed and discussed. By means of combining a tracks embedding and a titles embedding our recommender has reached 19 percent of accuracy. Our model has been created and trained using the MPD (million playlists dataset) given by Spotify as part of the RecSys Challenge 2018.

Resum

En aquest treball presentem un recomanador de cançons per a llistes de música. Mitjançant l'algorisme Word2Vec, una red neuronal poc profunda habitualment utilitzada per aprenetage de text, hem construït diversos embeddings utilitzant cançons i títols com a paraules d'un vocabulari inventat. Combinant un embedding de cançons i un de títols de llistes de música, i a través de diversos experiements combinant distància entre cançons i popularitat, hem arribat a aconseguir fins a un 19 per cent d'encert. El nostre model ha estat creat i entrenat a partir de l'MPD (dataset d'un milió de llistes de música) cedit per Spotify com a part del concurs RecSys Challenge 2018.

Resumen

En este trabajo presentamos un recomendador de canciones para listas de música. Mediante el algoritmo Word2Vec, una red neuronal poco profunda habitualmente utilizada para aprenetage de texto, hemos construido varios embeddings utilizando canciones y títulos como palabras de un vocabulario inventado. Combinando un embedding de canciones y uno de títulos de listas de música, y a través de varios experiements combinando distancia entre canciones y popularidad, hemos llegado a alcanzar hasta un 19 por ciento de acierto. Nuestro modelo ha sido creado y entrenado a partir del MPD (dataset de un millón de listas de música) cedido por Spotify como parte del concurso RecSys Challenge 2018.

Acknowledgements

I would like to thank Dr.Santi Seguí and Guillem Pasqual for guiding and helping me in this project. It has been a pleasure to work under their supervision.

I would also like to thank Oriol Pujol, Jordi Vitrià and specially Jose Mena for their collaboration on the Challenge.

And I would like to thank my parents for their constant support during these years.

Contents

1	Motivations	1
2	Introduction	2
2.1	The Problem	2
2.2	Recsys 2018	3
3	The Dataset	4
3.1	Stats	5
3.2	Playlist Features	5
3.3	Popular Items	6
3.4	Order	7
4	State of the Art	8
4.1	Collaborative Filtering	8
4.2	Similarity Based Algorithms	9
4.3	Statistical Models	10
4.4	Case Based Reasoning	10
4.5	Frequent Pattern Mining	10
4.6	Discrete Optimization	11
4.7	Hybrid Techniques	11
5	Neural Networks	12
5.1	Biological Neuron	12
5.2	Artificial Neuron	13
5.3	Activation Functions	14
5.4	The Network Topology	15
5.5	The Working of Neural Networks	16
5.6	Gradient Descent	17
5.7	Backpropagation	18
6	Word2vec	20
6.1	The idea	20
6.2	The Network Task	21
6.3	Model Details	21
6.4	The Hidden Layer	22
6.5	The Output Layer	23
6.6	Contexts	23
6.7	Improvements	24

7	Tracks Embedding	25
7.1	Model Characteristics	25
7.2	Assessing the quality of the model	27
7.3	Recommendation Algorithm	29
8	Titles Embedding	31
8.1	Model Characteristics	32
8.2	Assessing the quality of the model	32
8.3	Recommendation Algorithm	35
9	Results	36
9.1	Combining the models	36
9.2	Other improvements	37
9.3	Ranking	38
10	Conclusions	40
A	Appendix	42
	Bibliography	44
	List of Figures	45
	List of Tables	48

Chapter 1

Motivations

Deep learning has gained prominence in the last recent by out-performing traditional machine learning problems such as image classification and speech recognition. It is also noticeable that deep neural networks can be trained and experimented with by many, and not just few researchers at big tech companies and academia. It is, in brief, a wide open field and one of the most highly sought after skills nowadays. Considering the little I know about it, and how powerful it seems to be, I have decided to use my final degree project to learn as much as possible about this growing trend. My main goal is to understand and get to work with neural networks.

Even though recommenders were not part of my original plans, the Recsys Challenge 2018 has appeared to be a magnificent opportunity for my purposes. On the one hand, this challenge allows me to work with a clean dataset. It is large enough to make working models and light enough to train it with an average computer. It has a clear structure and different variables to take into account and to experiment. And most importantly, it has never been explored before. Understanding, analyzing and working with a dataset is also one of my goals.

On the other hand, the objective of this challenge is very clear: to recommend tracks to playlists. Nevertheless, the ways to achieve this are countless. To do an exploratory research of the state-of-the-art techniques regarding recommenders, and music recommendation systems is another goal I set. As well as choosing one of the methods and implementing it using my newly explored dataset.

Lastly, the fact that it is a challenge, makes this project all the more charming. Since I am in the motivations section, I will take this opportunity to say that apart from making a functioning music recommender based on machine learning, one of my strongest motivations is to be in the top 10 by the end of the challenge.

Chapter 2

Introduction

With the apparition of the internet and its ever-growing worldwide access, music consumption has taken a turn. On-line music listening platforms have emerged to be the new music provider. Thanks to cloud-based service like Spotify, the consumer has now instant on-demand access to millions of songs. In order to help users explore these large collections of music, music recommender systems play an important role. In fact, research in music recommender systems (MRS) has become a very popular topic in the recent years. Personalization is attractive both for content providers, who can increase sales or views, and for customers, who can find interesting content more easily.

2.1 The Problem

The problem that we are going to tackle, is the called task of playlist recommendation (PR). But what is a playlist? It is defined as a list of recorded songs or pieces of music chosen to be listened together. That means that there exists some kind of relation/s among the tracks for them to appear in the same playlist. The PR task consists on finding such relations and adding one or more tracks to a playlist in a way that fits the same target characteristics as the original playlist. That is the goal we are going to try to achieve in this work by means of understanding and using deep learning techniques on the given dataset.



2.2 Recsys 2018

The ACM Recommender Systems conference (RecSys) is the premier international forum for the presentation and discussion of new research results, systems and techniques in the broad field of recommender systems. Recommendation is a particular form of information filtering, that exploits past behaviors and user similarities to generate a list of information items that is personally tailored to an end-user's preferences. As part of the conference, Recsys organizes annually a competition.

This year's challenge focuses on music recommendation, specifically the challenge of automatic playlist continuation. It is organized by Spotify, an online music streaming service with over 140 million active users and over 30 million tracks. One of its popular features is the ability to create playlists, and the service currently hosts over 2 billion playlists. As part of this challenge, Spotify will be releasing a public dataset of playlists, consisting of a large number of playlist titles and associated track listings. The evaluation set will contain a set of playlists from which a number of tracks have been withheld. The task will be to predict the missing tracks in those playlists.

For more information about the challenge please refer to the RecSys Challenge 2018 website or to following paper recently published by the challenge organizers [1].



Chapter 3

The Dataset

As part of this challenge, Spotify has released the Million Playlist Dataset. It comprises a set of 1.000.000 playlists that have been created by Spotify users from US, and includes playlist titles, track listings and other metadata. In figure 3.1 we can see the overall demographics of users contributing to the MPD by gender and by age.

The MPD however, does not contain any information related to the users themselves, so any kind of user oriented recommendation system is out of the table.

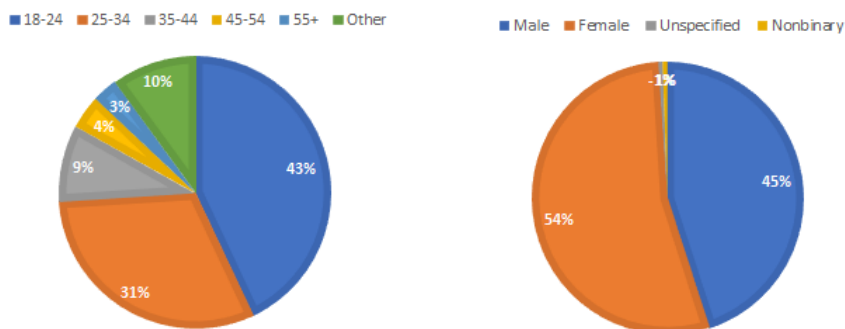


FIGURE 3.1: Demographics about Spotify users who originally made the MPD playlists. Age on the left, gender on the right.

Data visualization is one of the core skills in data science. In order to start building useful models, we need to understand the underlying dataset. We will try in the following pages to have a clear idea of the most relevant variables and get a first impression of the structure of the data.

3.1 Stats

First of all, we need to know what is the exact information we have and with how much information we are dealing with. From here, we will determine which attributes appear to be useful, and which irrelevant, and see if we do not have enough information and we need to create more (by crossing and mixing different attributes for instance), or maybe we have too much information, and some threshold needs to be applied to ignore part of it. A general summary of some relevant data can be found in table 3.1 where we can see that there are near 2.2 million unique tracks. Not any model can be this big, specially when it comes to neural networks. In some cases large datasets allow better training results, other times it just slows them down, but sometimes, too big of a dataset makes a problem impossible to solve. In any case, it might be worth considering only the tracks appearing more than n times to see what are our options. We can see what the impact of ignoring unpopular songs would be in figure 3.2.

number of tracks	66346428
number of unique tracks	2262292
number of unique albums	734684
number of unique artists	295860
number of unique titles	92944
number of playlists with descriptions	18760
number of unique normalized titles	15876
avg playlist length	66

TABLE 3.1: MPD stats.

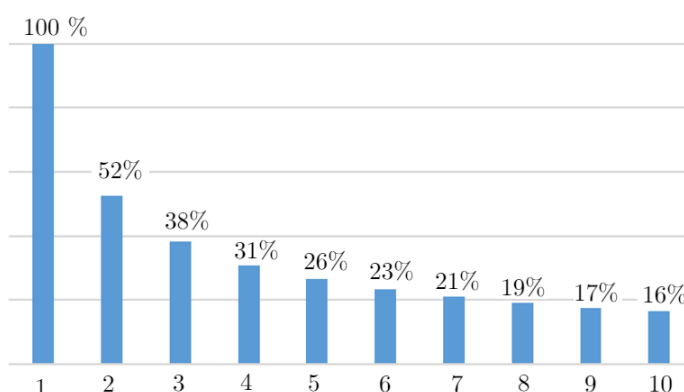


FIGURE 3.2: MPD percentage left when considering tracks appearing at least n times.

3.2 Playlist Features

As a matter of fact, the playlists have been selected to follow some criteria such as to have a minimum of 5 tracks and no more than 250, 3 different artist, 2 unique albums

and at least one follower. Within this gap, let us have a look at the most common features of the playlists. Looking at figures 3.3, we can tell that there is a clear tendency for playlists to have around 40 tracks, 15 albums, 10 artist and last about 2 hours. Not much information seem to carry the number of followers.

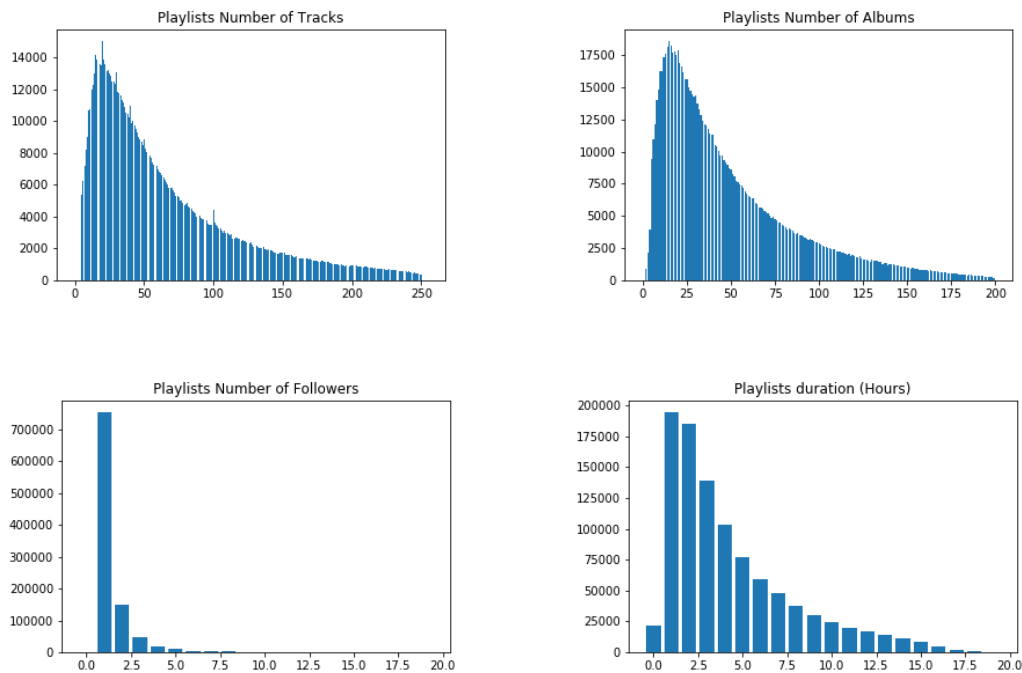


FIGURE 3.3: MPD histograms of number tracks, albums, followers and playlists duration in hours.

3.3 Popular Items

Looking at the popularity of tracks and playlists titles 3.4 (same happens with artists and albums), we can notice the so called “long tail structure”, which means that there is a portion of the distribution having a large number of occurrences far from the ”head” or central part of the distribution.

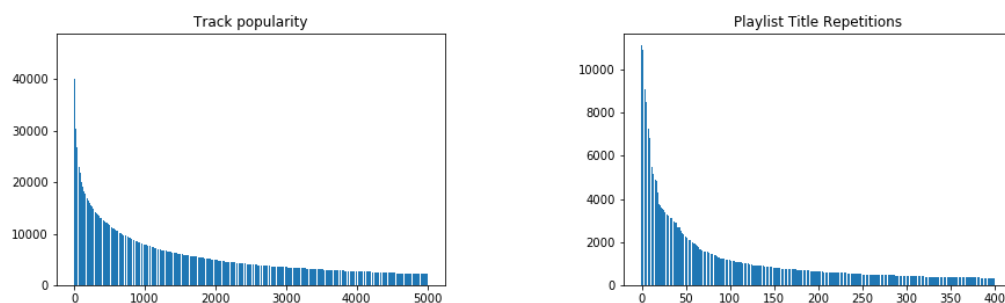


FIGURE 3.4: MPD tracks and playlist titles number of apparitions in the MPD.

3.4 Order

Lastly, having a look at the average position of tracks on the playlist they appear, and plotting it against their popularity, no relevant conclusion can be guessed, as we can see in figure 3.5. The order of the tracks seems to be random, at least for now.

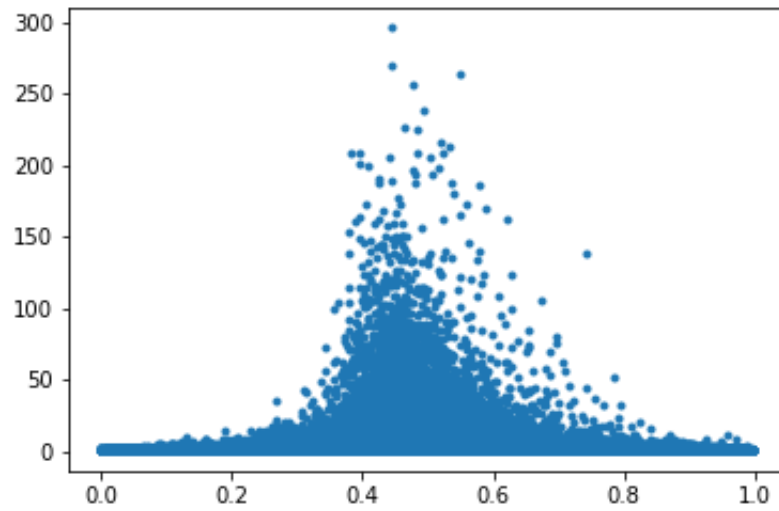


FIGURE 3.5: Tracks popularity vs average position on the playlists they appear.

Chapter 4

State of the Art

As exposed on the paper [2], there are different ways to approach the music recommendation problem, which depend mostly on two factors. First, the kind of information used to determine whether a playlist and its tracks satisfy the target characteristics. This information can be classified in the following categories: the audio signal, metadata, social web data, and usage data. And second, the strategy for playlist recommendation. In the next pages, we will review these strategies to have a solid background on the possible techniques to use later on in our own work.

4.1 Collaborative Filtering

Collaborative filtering, is a user oriented recommendation system that filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future. In the neighborhood-based approach a number of users is selected based on their similarity to the active user. A prediction for the active user is made by calculating a weighted average of the ratings of the selected users. However, in our case we do not have users. But by taking each playlist as a user, and its tracks as items the "user" likes, we could apply this approach and recommend items to a playlist by looking to similar playlists [3, 4].



FIGURE 4.1: Collavorative filtering recommends items to a user based on its similar peers' items.

4.2 Similarity Based Algorithms

These methods rely on the intrinsic features of tracks (the audio signal) rather than any data relation between users or playlist characteristics. Using acoustic-based similarity measure, a metric space is created. The dimensions of the tracks correspond to different features such as tempo or tone. A possible way to recommend is by making a model for each playlist and looking for the closest tracks in the space [5]. One of the benefits of this system, is that it is possible to measure some playlists features. For example, in [6], the authors measured the diversity of a playlist by calculating the volume of the playlists associated all-tracks-enclosing-ellipsoid in the space.

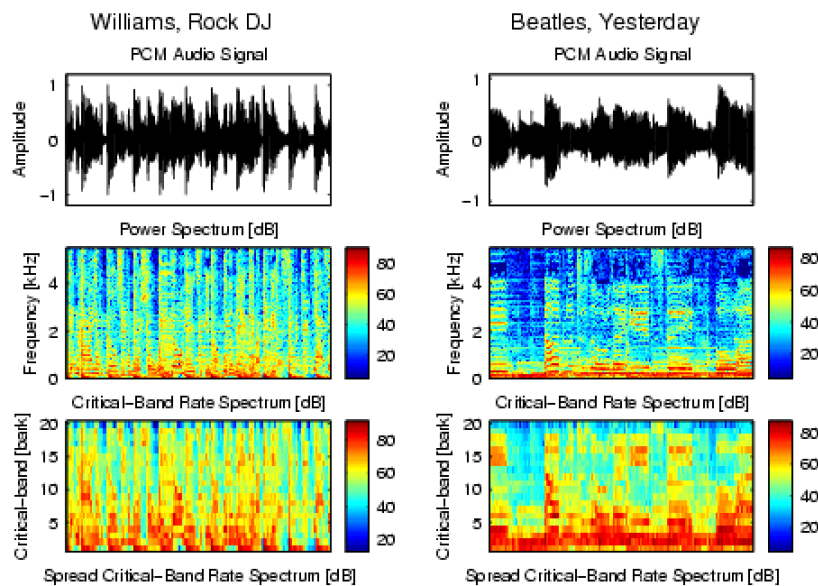


FIGURE 4.2: Audio signal comparison of two songs.

4.3 Statistical Models

Similarly to similarity based algorithms, statistical model tries to create a vector space out of the given items. In this case, however, we do not use the characteristics of each item, but the way it is related to the rest of items. There are several techniques to associate coordinates to an item by looking at its context. For instance in [7], the authors take each playlist as a list of strings belonging to some unknown language, and try to predict the next song as if it was a sentence they were trying to complete. In [8, 9], each track receives its coordinates by using a likelihood maximization heuristic. And in [10], the models select tracks based on distributions over tracks using latent clusters that were obtained by applying latent Dirichlet allocation.

4.4 Case Based Reasoning

The general idea of case-based reasoning techniques is to exploit information about problem settings (cases) encountered in the past to solve new problems. We can find an example of that in the article [11]. The case base in their scenario consists of a set of playlists created by a user community. Within these playlists, frequent subsequences (patterns) are identified. In contrast to typical case-based reasoning approaches, the goal of their work is not to find the most similar playlists given some seed track, but to find those considered to be the most “useful,” for example, in terms of diversity. The elements of the retrieved playlists are then combined to generate a new playlist for a given seed track.

4.5 Frequent Pattern Mining

The search of association rules are often applied for shopping basket analysis problems and have the following form: considering A and B to be sets of items, the rule we are looking for are “whenever A was bought, also the items in B were” [12]. This method applied to our problem, would consist on finding global patterns on the co-occurrence of tracks in a playlist. We can find two examples of frequent pattern mining techniques applied to playlist recommendation in [8], where Markov chains are used, and in [13], where n-grams are used.

4.6 Discrete Optimization

Given a set of tracks, their characteristics, and a set of explicitly specified constraints capturing the desired characteristics, the goal is to create one arbitrary or an optimal sequence of tracks that satisfies the constraints. In the paper [14] the task is taken as a constraint satisfaction problem (CSP) and apply the existing search algorithms to select sequences of tracks that satisfy various constraints, in that case from a comparably small catalogue. [15, 16] the problem is modeled as an integer linear program.

4.7 Hybrid Techniques

Hybridization is often used to combine the advantages of different techniques and at the same time avoid the drawbacks of individual techniques. There are different combination methods such as classification, [17], weighted hybridization [7] or multicriteria kNN-based collaborative filtering.

Chapter 5

Neural Networks

As we have seen, there are many ways to tackle the playlist recommendation problem. But let us remember that the original purpose of this work was not to solve this challenge, but rather to learn about neural networks. And that is what we are going to do before getting into detail on the practical solution of our problem (which is -spoiler alert- a practical application of a neural network).

5.1 Biological Neuron

The human brain is a highly complicated machine capable of solving very complex problems. Although we have a good understanding of some of the basic operations that drive the brain, we are still far from understanding everything there is to know about it. A neural network (NN) is a network consisting of connected neurons that can fire electric pulses through one another. These connections are possible thanks to the brain cell's four main parts (see in figure 5.2):

1. Dendrites: accept inputs (electric impulses).
2. Soma: processes the inputs.
3. Axon: Turn the processed inputs into a form that can be accepted by the next neuron i.e. converts processed inputs into output.
4. Synapses: The electrochemical contact between neurons. Using synapses, a neuron can transfer the outputs of that neuron to the inputs of the next neuron.

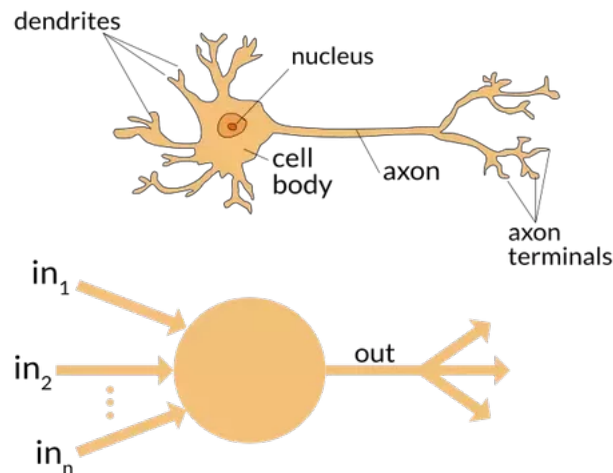


FIGURE 5.1: Structure of a biological neuron.

When a neuron receives enough electric pulses through its dendrites, it activates and fires a pulse through its axon, which is then received by other neurons. In this way information can propagate through the NN. The connections change throughout the lifetime of a neuron and the amount of incoming pulses needed to activate a neuron (the threshold) also change. This behaviour allows the NN to learn.

5.2 Artificial Neuron

It is not possible (at the moment) to make an artificial brain, but it is possible to make simplified artificial neurons and simplified artificial neural networks (ANN) by creating a similar data processing structure. ANNs are not intelligent, but they are good for recognizing patterns and making simple rules for complex problem. For instance, an ANN trained on images of different animals is able to predict whether an animal outside the original set is a cat or not. This is a very desirable feature of ANNs, because you do not need to know the characteristics defining a cat, the ANN will find out by itself (in this work, we will only consider supervised learning).

Artificial neurons are extremely simplified versions of biological neurons. The signal at a connection between artificial neurons is a real number, instead of an electric pulse, represented by x_i . Each input is “processed” i.e., multiplied by some weight w_i , and all $w_i x_i$ products are added. After that, the processed input is passed through a function, known as the “activation function”, which converts the processed input to output. So, following the analogy, the artificial neuron four main parts are (see in figure 5.2):

1. Inputs: numerical values (x_i).
2. Weights: changing numerical attributes of a neuron (w_i). This changing is determined by the learning algorithm, which we will study later on.
3. Activation function: transforms the input into an output. Different functions serve different purposes, we will look at them in a subsequent section.
4. Output: becomes the input of another neuron.

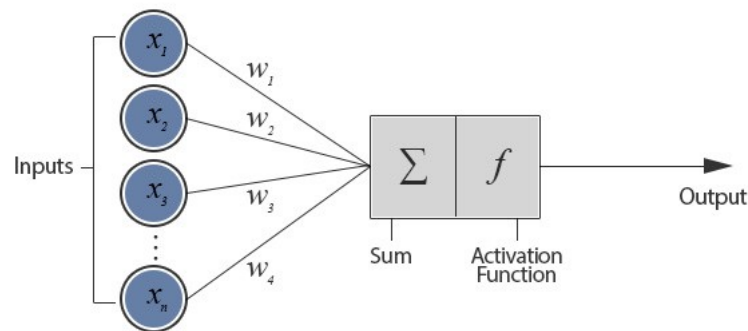


FIGURE 5.2: Structure of an artificial neuron.

5.3 Activation Functions

The general mathematical definition of an implemented neural neuron is, as briefly mentioned above:

$$y(x) = g\left(\sum_{n=1}^{\infty} w_n x_n\right)$$

The activation function g weights how powerful the output (if any) should be from the neuron, based on the sum of the input. There are several kinds of activation functions which serve different purposes. It is a very important, when making a network, to choose the right function. In table 5.3 we can see several examples of functions, as well as its derivatives. For reasons we will explain later, it is important that an activation function is differentiable.










Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

FIGURE 5.3: Activation function examples.

5.4 The Network Topology

Artificial neurons are aggregated into layers and the union of several layers make a neural network. The connections among neurons within a layer, and the connections between layers configure the topology of the network. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Many characteristics of the network will depend on these connections, so it is important to know the different kinds and their implications. These are the most relevant features:

1. Layer dimensions: from input to output, each layer's dimension can range from one to thousands, usually changing from one layer to the next one. Whether from one layer to another the dimension increases, decreases, stays the same, or increases and decreases (auto encoder) it is important, to respect the changing proportion in order for the network to work properly (usually following a logarithmic scale).
2. Number of layers: Adding more layers (usually) increases the accuracy of the network, but it also increases the cost, and if there are too many, some of them might become useless.

3. Connectivity: layers can be fully connected (all the neurons in each layer are connected to all the neurons in the next layer) which is the usual case, as well as the most cost effective. Layers can also be partially connected. Also, layers can be connected in a chronological order, or go back and forward, or skipping layers in some cases in order to avoid over-fitting (over-fitting is giving precise results for the training data, but incorrect results for all other data).

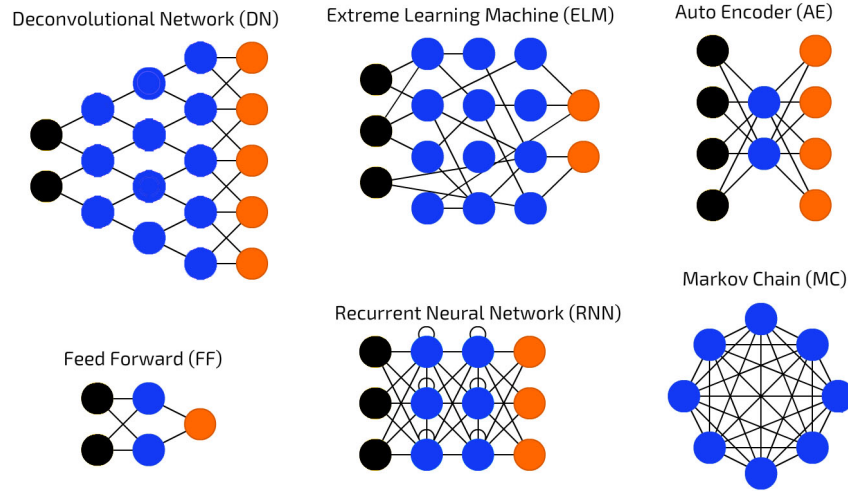


FIGURE 5.4: Neural network different structures.

5.5 The Working of Neural Networks

We have briefly mentioned that an ANN learns by changing the weights. But how does this changing affect the results? I like to think of it as a kind of Galton machine (see 5.5), in the sense that the input is a piece of data (blue ball), that after going through neurons with well-adjusted weights (orange dots) ends up in the right output category.

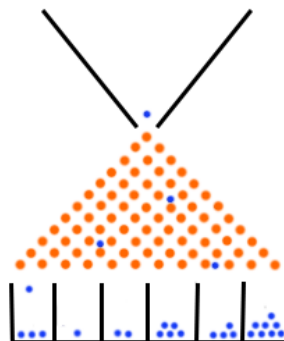


FIGURE 5.5: Galton machine.

But before the neural network can accurately predict the output, it needs to be trained on some data, usually consisting of input-output pairs. We wish to adjust the weights in the ANN, to make the ANN give the same outputs as seen in the training data. The “training” phase starts by randomly initializing all weights (i.e. weights associated with each artificial neuron). Then, inputs are fed to the network, activations of all nodes in layers are calculated, and finally, we get the neural network output. This output can then be evaluated by comparing it to the expected output, and the error calculated. The function that measures the error is known as the “cost function”. Hence the training process can be seen as an optimization problem where the aim is to adjust the weights in order to minimize this cost function.

It is worth mentioning that there are different cost functions, and the choice for each case depends on various factors such as the activation function. Clearly a multiclass classification network should be evaluated differently than a binary classification network. The technique used to minimize the cost function is called “gradient descent”.

5.6 Gradient Descent

The idea is that we make small steps in the direction of the gradient, and we hope that eventually, we’ll be at the global minima. However, that would be the case only if the plot is convex. Usually, the plot is not perfectly convex, resulting in a few local minima (5.6). For now, we’ll assume that the local minima are good approximations of the global minimum (which is usually the case). So, we make small updates on the weights, each time moving them in the direction of the gradient. We multiply the updates with a parameter, known as “learning rate”.

To compute the gradient of the loss function with respect to the weights in the ANN, an algorithm called Backpropagation is used.

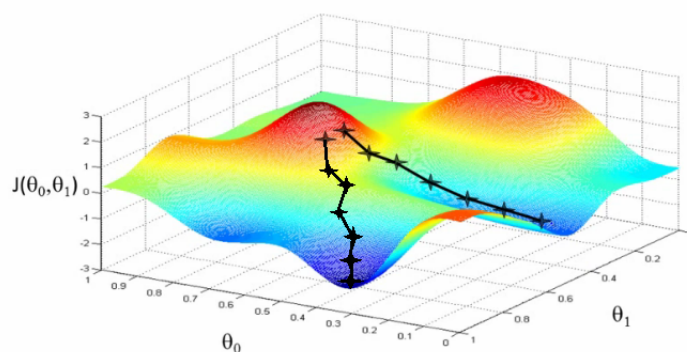


FIGURE 5.6: The plot of cost function vs weight is more or less convex and looks something like this. In black we can see the steps towards the local minima.

5.7 Backpropagation

The backpropagation algorithm works in much the same way as the name suggests: After propagating an input through the network, the error is calculated and the error is propagated back through the network while the weights are adjusted in order to make the error smaller. Although we want to minimize the mean square error for all the training data, the most efficient way of doing this with the backpropagation algorithm, is to train on data sequentially one input at a time, instead of training on the combined data.

First the input is propagated through the ANN to the output. After this, the error e_k on a single output neuron k can be calculated as:

$$e_k = d_k - y_k$$

Where y_k is the calculated output and d_k is the desired output of neuron k . This error value is used to calculate a σ_k value, which is again used for adjusting the weights. The σ_k value is calculated by:

$$\sigma_k = e_k g'(y_k)$$

Where g' is the derived activation function. The need for calculating the derived activation function was why we expressed the need for a differentiable activation function earlier.

When the σ_k value is calculated, we can calculate the σ_j values for preceding layers. The σ_j values of the previous layer is calculated from the σ_k values of this layer. By the following equation:

$$\sigma_j = \eta g'(y_j) \sum_{k=0}^K \sigma_k w_{jk}$$

Where K is the number of neurons in this layer and η is the learning rate parameter, which determines how much the weight should be adjusted. The more advanced gradient descent algorithms does not use a learning rate, but a set of more advanced parameters that makes a more qualified guess to how much the weight should be adjusted.

Using these σ values, the Δw values that the weights should be adjusted by, can be calculated by:

$$\Delta w_{jk} = \sigma_j y_k$$

The Δw_{jk} value is used to adjust the weight w_{jk} , by $w_{jk} = w_{jk} + \Delta w_{jk}$ and the backpropagation algorithm moves on to the next input and adjusts the weights according to the output. This process goes on until a certain stop criteria is reached. The stop criteria is typically determined by measuring the mean square error of the training data while training with the data, when this mean square error reaches a certain limit, the training is stopped. More advanced stopping criteria involving both training and testing data are also used.

Chapter 6

Word2vec

Once given the notion of what a neural network is, let us focus on the specific kind that has been used in this work. Word2vec is a shallow two-layer neural network trained to reconstruct linguistic context of words. In fact, it is a clear example that trading complexity for efficiency can produce great results, since a simpler model gains the ability to learn from much bigger datasets.

But what does word2vec do? Given a large corpus of text, this method produces a vector space, typically of hundreds of dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. The idea is that words sharing common contexts in the corpus are located close to one another in the space.

6.1 The idea

Word2Vec uses a trick very common in machine learning, and the basis of auto-encoders. As seen in figure 6.1 the autoencoder's structure is focused on compressing and decompressing the input data. By training the network to recompose the input from a lower dimension, it aims to extract the relevant data features in a much-reduced piece of data.

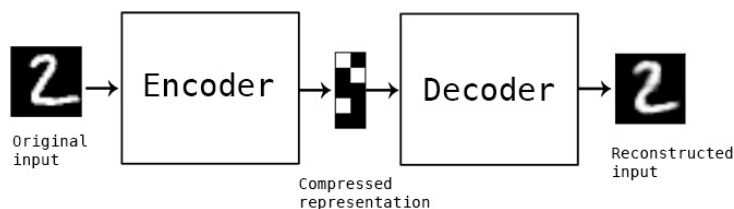


FIGURE 6.1: Autoencoder schematic functioning.

Word2vec trains a simple neural network with a single hidden layer to perform a certain task but ends up not using the network for this task. Instead, the goal is actually just to learn the weights of the hidden layer which will be the “word vectors” that it aims to learn.

6.2 The Network Task

What the network is trained to do is the following: given a specific word in the middle of a sentence (the input word), look at the words nearby and pick one at random. The network is going to tell us the probability for every word in the vocabulary of being the chosen “nearby word”. The output probabilities are going to relate to how likely it is to find each vocabulary word nearby our input word. For example, given the word “mathematics”, the output probabilities should be much higher for words like “algebra” and “probabilities” than unrelated words like “lipstick”. The network is fed word pairs found in the corpus to train. The way these pairs are taken is shown in the example below, 6.2. The network is going to learn the statistics from the number of times each pairing shows up.

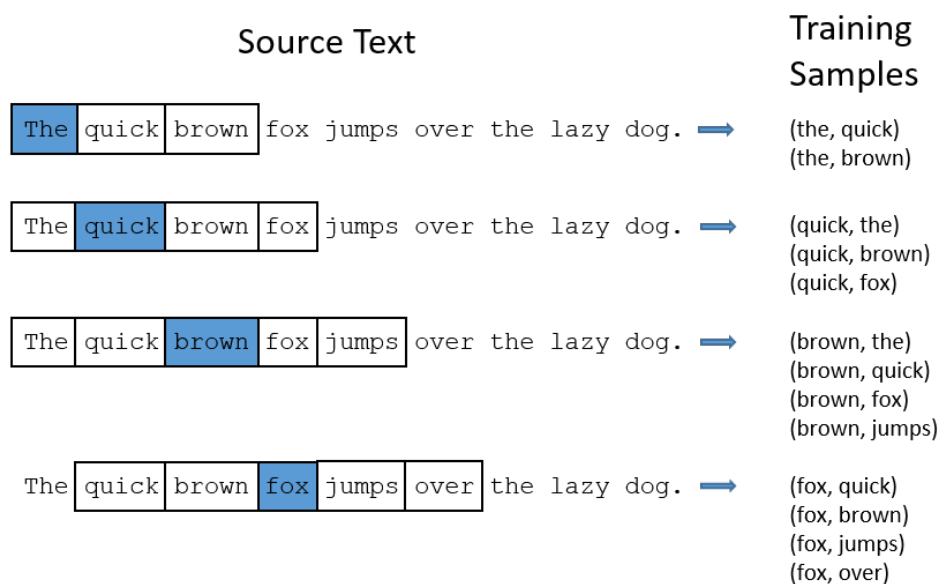


FIGURE 6.2: Word pairs taken from a sentence using a window size of 2. The word highlighted in blue is the input word.

6.3 Model Details

But what exactly are the input and the output of the network? Words cannot be fed just as a text string, it must be numerical, all same sized data. So, what is done, is to

convert each word into a one-hot vector (which is a vector with a single 1 and the rest of the values to 0) of size the length of the vocabulary (let's say 10,000). The output of the network is a single vector (also with 10,000 components) containing, for every word in our vocabulary, the probability that a randomly selected nearby word is that vocabulary word. Here's the architecture of our neural network.

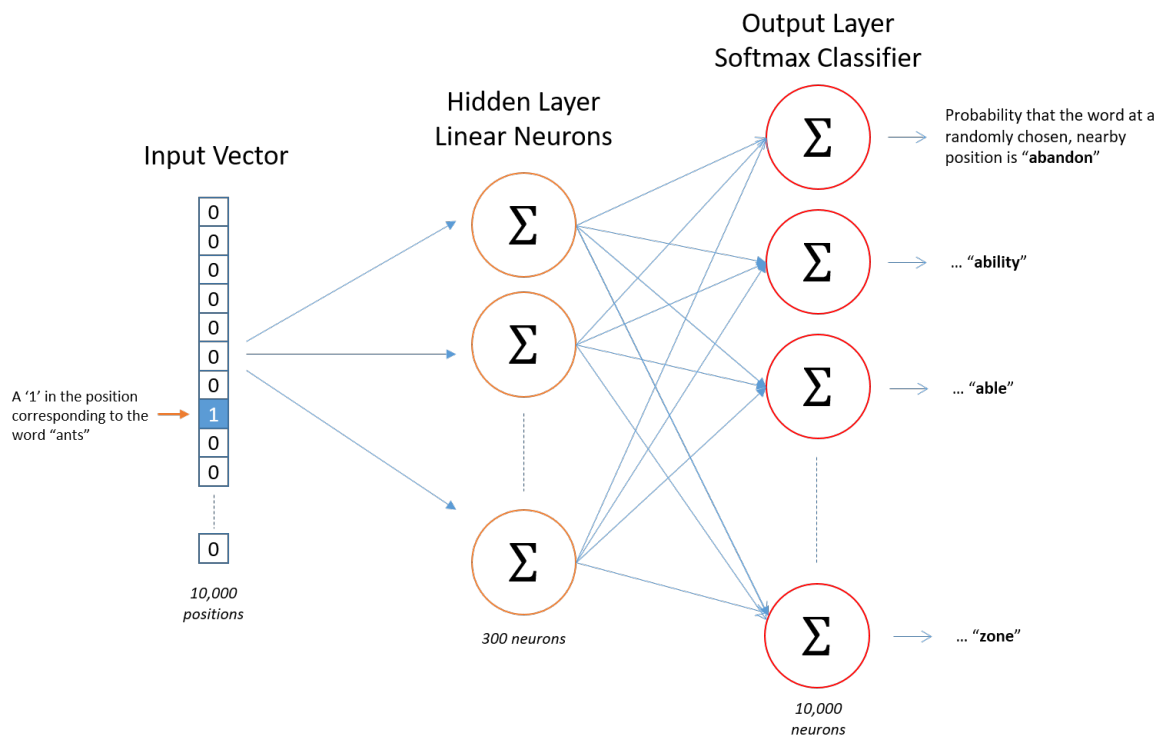


FIGURE 6.3: Neural network with one-hot vector input word and one-hot vector output. But when evaluated on an input word, the output vector will actually be a probability distribution (i.e., a bunch of floating point values, not a one-hot vector).

There is no activation function on the hidden layer neurons, but there is on the output neurons, and this function is called softmax. We'll come back to this later.

6.4 The Hidden Layer

Let us consider word vectors with 300 features. The hidden layer is going to be represented by a weight matrix with 10,000 rows (one for every word in our vocabulary) and 300 columns (one for every hidden neuron). These are actually what will be the word vectors. Hence, the end goal of all of this is really just to learn this hidden layer weight matrix – the output layer we'll just toss when we're done. Let it be noticed that by using one-hot vectors as an input, each word training will effectively just select the matrix row corresponding to the "1". Here's a small example to give you a visual.

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

FIGURE 6.4: Example of how the output is only affected by the corresponding part of the hidden layer - or a simple matrix multiplication.

This means that the hidden layer of this model is really just operating as a lookup table. The output of the hidden layer is just the “word vector” for the input word.

6.5 The Output Layer

The output layer is a softmax regression classifier. Briefly explained, it means that the output neuron will produce an output between 0 and 1, and the sum of all these output values will add up to 1. Specifically, each output neuron has a weight vector which it multiplies against the word vector from the hidden layer, then it applies the function $\exp(x)$ to the result. Finally, in order to get the outputs to sum up to 1, we divide this result by the sum of the results from all 10,000 output nodes. Here’s an illustration of calculating the output of the output neuron for the word “car”.

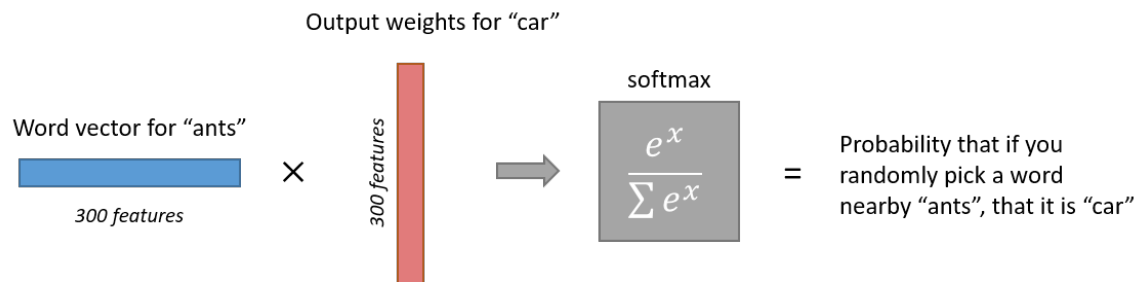


FIGURE 6.5: Word2vec processing of a word.

6.6 Contexts

If two different words have very similar “contexts”, then our model needs to output very similar results for these two words. And one way for the network to output similar context predictions for these two words is if the word vectors are similar. So, if two words have similar contexts, then our network is motivated to learn similar word vectors for these two words. And what does it mean for two words to have similar contexts? I think you could expect that synonyms like “intelligent” and “smart” would have very similar contexts. Or that words that are related, like “engine” and “transmission”, would

probably have similar contexts as well. This can also handle stemming for you – the network will likely learn similar word vectors for the words “ant” and “ants” because these should have similar contexts.

6.7 Improvements

Skip-gram neural network contains a huge number of weights. Considering 300 features and a vocab of 10,000 words, that’s 3M weights in the hidden layer and output layer each. Training this on a large dataset would be prohibitive, so the word2vec authors introduced a number of tweaks to make training feasible. These improvements do not only reduce the compute burden of the training process, but also improved the quality of their resulting word vectors as well. These three innovations, presented in [18] are:

1. Treating common word pairs or phrases as single “words” in their model. For example “Boston Globe” (a newspaper) has a much different meaning than the individual words “Boston” and “Globe”. So it makes sense to treat “Boston Globe”, wherever it occurs in the text, as a single word with its own word vector representation.
2. Subsampling frequent words to decrease the number of training examples. Words such as “the” appear in the context of pretty much every word. Word2Vec implements a “subsampling” scheme to address this. For each word we encounter in our training text, there is a chance that we will effectively delete it from the text. The probability that the word is cut is related to the word’s frequency.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights. This avoids adjusting all of the neurons for each training example.

For the most detailed and accurate explanation of word2vec check [19, 20], read the word2vec authors paper [21] or a simplified version [22].

Chapter 7

Tracks Embedding

Once observed the different techniques to approach the playlist recommendation problem, and now that we know what neural networks are and how Word2vec works, we can proceed to explain our method. You might have been wondering what does a text recommender like word2vec have to do with music, but the fact is that word2vec is not solely applied to text corpus. It has been successfully applied to biological sequences, and in our case, to music playlists.

Considering each track as a word, and each sequence of tracks (playlist) as a sentence, one can build a vocabulary made of songs. In this way, if two tracks appear in very similar contexts (for example Christmas songs usually appear all together), then the model will output very close vectors in the space. That is the idea behind our recommendation system.

7.1 Model Characteristics

As mentioned above, the vocabulary used for this model are tracks, and the way they have been grouped are by playlists. All tracks have been given a numeric id, sorting them by popularity. So number 1 corresponds to the most popular track in the MPD, 2 to the second most popular, and so on. Only tracks appearing 5 times or more in the MPD have been taken into account in this specific model, so the vocabulary contains nearly 600.000 words.

The word2vec architecture selected is skip-gram. This model uses the current word to predict the surrounding window of context words. There is another possible word2vec structure named CBOW, which predicts the current word from a window of surrounding

context words. Even though CBOW is a faster model, Skip-gram has shown to produce better results in the long term. The dimension of the vectors created is 300, and the window size for training is 5.

The complete training corpus is 1.000.000 sentences (one for each playlist of the MPD), and for each training iteration, all tracks of each sentence have been shuffled. This serves the purpose of letting each track be related to every other track in the playlist by appearing close to it in some of the shuffled iterations. By doing that, we are tossing any possible relation of order between songs, but our hypothesis is that the order in which the tracks are ordered is rather irrelevant.

In figure 7.1 we can see an example of how a playlist is selected and transformed before it is fed into our model to train. This process is done for all the 1.000.0000 playlists of our dataset.

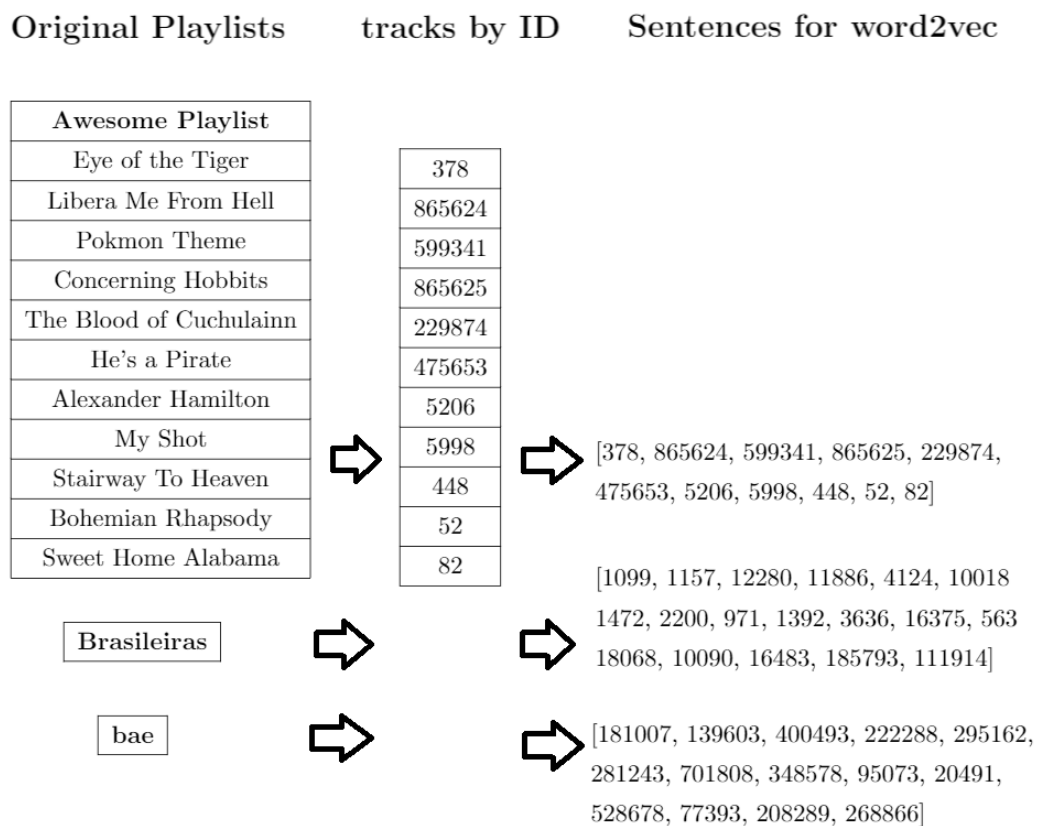


FIGURE 7.1: Converting playlists to sentences to train our word2vec model.

7.2 Assessing the quality of the model

It is logical to think that rap songs do not usually appear together with classical music songs. And the same should happen with different genres of music. So, one could expect some clusters should easily appear in our embedding space, and that would mean that our model is working properly.

To plot our 300 dimensional space embedding, we have reduced it down to 20 dimensions using PCA and plot it in 2 dimensions by means of T-SNE [23] which is an algorithm specialized on plotting high dimensional spaces, keeping its most important characteristics.

Also, in order to have track samples of different music genres, we have selected the 100 most common tracks within playlists of a specific title. For example, out of playlists named “Christmas”, the most times appearing songs are *All I Want for Christmas Is You* and *It’s Beginning To Look A Lot Like Christmas*. In this way, we have selected 7 categories and plot them among other 2000 random tracks. We can see in figure 7.2 how each category forms a cluster. These clusters are good indicator that our embedding is working.

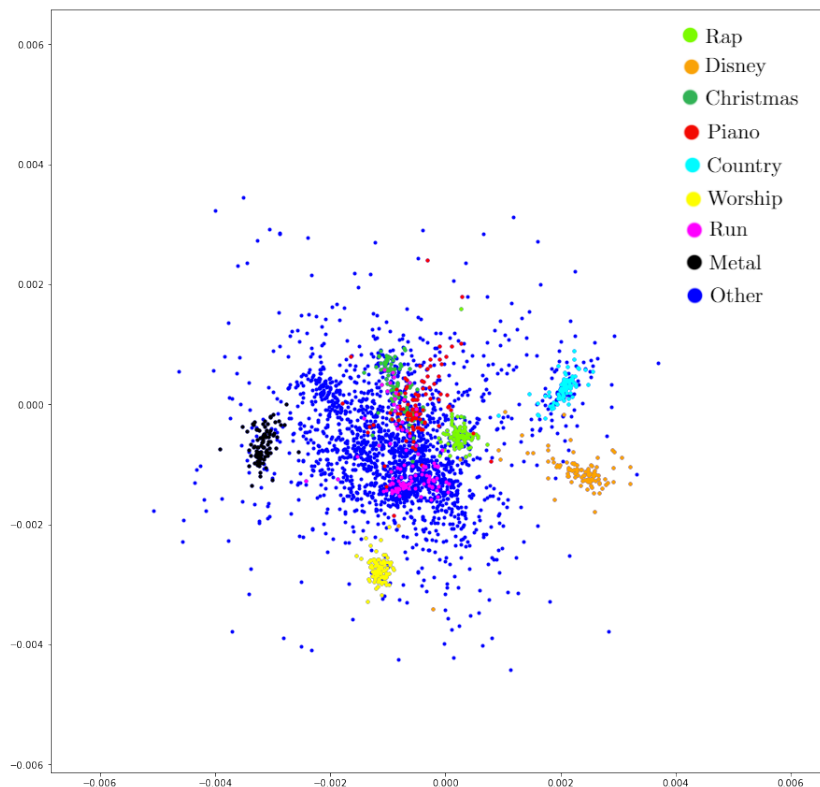


FIGURE 7.2: Word2vec tracks embedding t-SNE representation. 100 tracks per category + 2000 random tracks. Perplexity = 150. Learning rate = 30.

Changing the number of random tracks appearing in the plot, or modifying t-SNE parameters such as the perplexity do not affect the clusters formation as seen in 7.3. So, even though the t-SNE is an approximation of the real 300 dimensional space, it gives us a good idea of how the real space is and tells us there is a very high possibility that the embedding will serve our purposes.

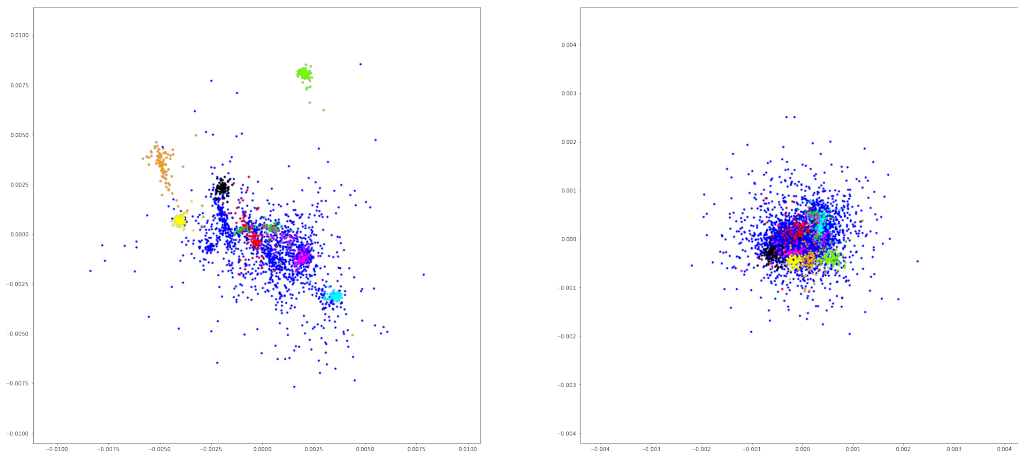


FIGURE 7.3: 100 tracks per category + 1500 tracks and 3500 tracks respectively. Perplexity=150. Learning rate = 30.

Even though these plots are very promising, it is not enough to be certain that the embedding is working and, needless to say, it does not tell us how good a result it would produce. Let us remember that our aim is to predict tracks for incomplete playlists. More specifically, we need to recommend 500 tracks to each playlist of the **challenge set**. The challenge set consists of 10.000 playlists with the following characteristics:

Number of playlists	Tracks per playlist	Titles	Tracks position
1000	0	Yes	-
1000	1	Yes	First
1000	5	Yes	First
1000	5	No	First
1000	10	Yes	First
1000	10	No	First
1000	25	Yes	First
1000	25	Yes	Random
1000	100	Yes	First
1000	100	Yes	Random

TABLE 7.1: Challenge set playlists features.

In order to test our recommendations we created a fake challenge set (the fake set) with the same characteristics as the challenge set. We made it by taking playlists of the MPD and erasing some of its tracks/titles. This way we can compare the recommendation tracks with the original tracks and evaluate how many tracks we guessed right.

It is important to notice, that our models were trained with the whole MPD instead of splitting it into train and test as it is usually done. That is due to the fact that we consider the MPD the train set and the challenge set the test set. But we do not know the real solution of the challenge set, and only one submission can be evaluated every day. That narrows down our number of tests, and that is why we created the fake set. But by creating it out of playlists of the MPD (the train set), the results will be irremediably better than they actually are. That is not really a problem by itself, since it will affect all playlists equally, but it is important to keep it in mind.

7.3 Recommendation Algorithm

Once we have a metric space, it is fairly easy to make recommendations. There are several possible ways to approach the problem. One of the most used ones, and as effective as it is simple, is to use the average position of a playlist and recommend the closest 500 tracks. This procedure can be seen in 7.4, where in red we have highlighted all the tracks of a playlist in our space. The average position of these tracks is computed (in yellow), and the closest 500 tracks to this point are selected (in green).

We have tried other heuristics such as clusterizing these playlist tracks into 3 groups with K-Menas and recommending the closest 170 tracks to each of the centroids, but it produced worse results.

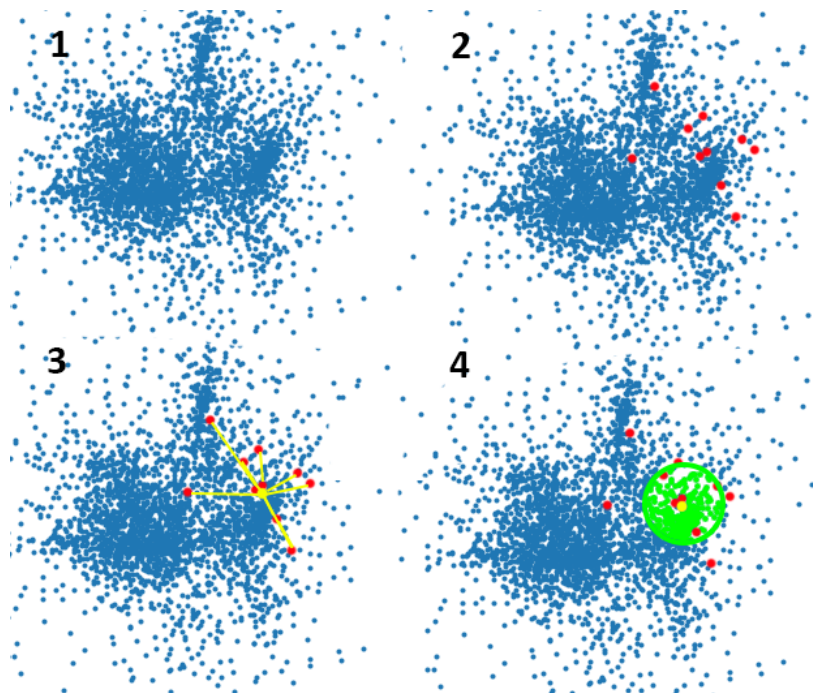


FIGURE 7.4: Process of recommending tracks to a playlist using the tracks embedding.

It is no surprise that the method produces different results depending on how many tracks the given playlist contains. We can see in the following table the accuracy of the model evaluated on the fake set for different kinds of playlists.

number of seeds	accuracy
1	32 %
5	43 %
10	45 %
25	44 %
100	38 %

TABLE 7.2: Rrecommending performance for different number of seeds playlists.

Chapter 8

Titles Embedding

A noticeably important feature of the MPD is the playlists title. There is a strong relation between the tracks of a playlist and its title. It makes a lot of sense, because the title is usually a very good summary of what kind of music the playlist contains. After all, playlists are a way to organize music, and if a user wants to be able to navigate through his library, he'll need to label his playlists in a clear and logical way.

Looking at the playlists' titles gives a very good idea on what users tend to group their playlists by. Here are some examples of different features users most commonly consider when creating a new playlist.

genre	country	rap	classic rock	hip hop	achoustic
place	gym	library	volleyball	train	beach
activity	homework	run dance	sleep	study	yoga
emotional state	happy	calm	chill	inspiration	nostalgia
special event	party	wedding	baby shower	birthday	new year
specific time	good morning	cocktail hour	christmas	suumer 17	halloween
time peri- ods	80's	oldies	modern	youth	freshamn year
region	africa	latin	mexican	korean	ranchera
someone	mom	us	family	alex	baby

TABLE 8.1: Features by which some playlists are made, and 5 examples of each.

So, it would be very interesting to be able to predict tracks, only using its title. The first step towards that, and considering the title of this work, is to create an embedding of titles.

8.1 Model Characteristics

In this case, there is no such an obvious relation between the titles of playlists as there was between tracks, however, all playlist titles have something in common: its tracks. So, by putting upside down the previous relation, we will now consider all the playlists a track appears in and call it a sentence.

It is worth mentioning, that titles have been grouped by after slightly normalizing them. By lowering all letters, deleting irrelevant symbols and taking into account some writing practices such as 2017 = 2k17, we have been able to narrow down the number of different titles from 93.000 to 15.000. In table 8.2 we can see 5 examples of titles classified under the same category.

country	christma	run	mom	chil
not country	CHRISTmas	RUNNING	moms	Chilling.
not country	Christmas	Running	Mom	chilled
Country ;3	C H R I S T M A S	Run	mom.	chill?
country	christmas;))))	run	mom	chill
C O U N T R Y	christmas	run ²	For Mom	CHILLLLL
Country 1	Christmas!!!	Run.	Your mom	chill
country	It's Christmas!	RUN	Moms	chill
c o u n t r y	Christmas	Running	Mom	CHILLED
Country	CHRISTMAS!	*running	Mom	chilllll
Country	Christmas	Runnnn	mom	chilllllllll

TABLE 8.2: Playlist titles normalization examples.

For that reason, our embedding of titles has a vocabulary of 15.000 words approximately. The corpus in this case, contains near 2.000.000 sentences, one for each unique track. Shuffling is applied on the sentences after each iteration for in this case the order is completely irrelevant. The window, just as before, is set to 5, and the minimum number of appearances of a title is 1: we want all known titles to appear in our model. The dimensions of the vectors has been set at a hundred due to the fact that the vocabulary was significantly smaller than in the previous case, and less features need to be kept in order create a functional embedding.

8.2 Assessing the quality of the model

In this case, as the items we are working with are very understandable and intuitive, we can look at some examples to see if the model outputs make sense. In the following tables 8.3 we can see nine playlist titles and its top 10 most similar titles, as well as their similarity to the original title (how close in the embedding, vectors are from one another).

Looking carefully to several hundred titles and their closest results, it gives a strong impression the model works. It is able to notice features within the playlists such as the above mentioned genre, emotional state, activity... but not only that, it can abstract much more, as we will proceed to analyze.

Gender related titles such as *reggeton*, *hip-hop* or *acoustic*, present the highest resemblance with their neighbours. That is not shocking, since most songs do have a defined, preset gender that the users has in mind in this case when making the playlist. Even seasonal, or one-time events such as *Christmas*, *Halloween* or *wedding* perfect results are not that surprising considering that a lot of these songs were created with the purpose of being listened on these dates.

However, it is remarkable how the model is able to find relations in playlists named after supposedly subjective titles such as *Mom*, or *my favourites*. It is in cases like these ones that we can see how machine learning can outperform human action by finding relations that escape to our capacities. What I mean, is that given rock songs, any person who has a minimal musical culture, knows that they belong to rock genre. However, given a set of tracks that we don't know belong to a playlist called *Mom*, no one would guess they are "mum genre songs". And here lies the pattern finding magic of big data.

Another important fact is that our model does not only produce good results for most common, and highly appearing titles in the model. In tables 8.3, near the playlist name we can find its ID, which also relates tot the title popularity (low number are very frequent titles, high numbers are uncommon titles). Looking at the last tables, to an uncommon title name such as *flogging molly* we can see how good the results are. Flogging molly is an Irish pub music style, which matches very well with our model recommendations.

workout	4	spanish	53	summer 2015	99
workout music	0.77	spanish music	0.88	march 2015	0.66
workout1	0.74	latino music	0.86	summer 15	0.66
work out	0.73	spanish jam	0.85	2015	0.66
beast mode	0.72	spanish hit	0.85	spring 2015	0.63
gym tune	0.7	spanish vibe	0.84	june 2015	0.63
leg dai	0.7	cancion espanola	0.83	septemb 2015	0.63
kickbox	0.7	spanish playlist	0.82	fall 2015	0.62
boot camp	0.7	latin hit	0.81	februari 2015	0.62
0	.69	latin	0.81	august 2015	0.61
crossfit	0.69	hispan	0.81	2015 song	0.61
mom	115	my song	107	wedd ceremoni	800
mom music	0.66	all song	0.51	ceremoni music	0.85
mom dad	0.61	my music	0.47	befor ceremoni	0.81
mom song	0.6	top song	0.46	ceremoni	0.79
papa	0.59	fat	0.45	pre ceremoni	0.79
wedd danc playlist	0.59	theme	0.45	wedd music	0.74
anniversari	0.59	new life	0.45	ceremoni song	0.71
mom playlist	0.58	fav song	0.44	ludovico einaudi	0.71
great dai	0.58	gab	0.44	wedd dinner	0.71
mama	0.57	my stuf	0.43	piano song	0.69
nora	0.56	like these song	0.43	recept playlis	0.68
flogg molli	14004	eagl	2800	panti dropper	4000
irish song	0.87	roadtrip music	0.66	sexi music	0.62
irish pub	0.86	70 rock	0.66	sex time	0.6
irish drink song	0.85	rock.clas	0.65	bedroom	0.59
irish rock	0.85	billi joel	0.65	slow jam	0.58
st patrick dai	0.83	classic rock	0.65	0 ₀	0.57
irish folk	0.82	70	0.64	sweet love	0.57
irish	0.82	old rock	0.64	babi make	0.57
irish music	0.8	no1	0.63	sex music	0.56
ireland	0.79	elo	0.63	bedroom jam	0.56
paddi	0.77	easi rock	0.62	after hour	0.56

TABLE 8.3: 9 playlists titles, their ID, and its top 10 most similar playlist titles according to the titles embedding.

8.3 Recommendation Algorithm

Titles seem to relate nicely to one another, but, how is this useful to our tracks recommendation task? We need to associate to each title the songs that usually go with it. What we have done is the following: for each normalized title, we have selected all the playlists with this title, and taken all its tracks. Sorting them by number of occurrences, we have picked the most common 1000. In this way, we have built a list of titles and its associated 1000 tracks.

The way of recommending tracks to a given title is by looking into this table. But the point of the embedding is that we can relate a title with its most similar titles, so when given the title *spanish*, instead of looking only to the 1000 tracks of *spanish*, we will also look to *spanish music* and *latino music* among others. The number of playlists taken into account depends on the case, and to select the best 500 tracks among all these, we consider the total number of appearances and multiply each case with the similarity to the original title.

Lastly, to test our recommendation results, we have created a special list of the most common 1000 tracks per title, by only iterating over 90 % of the MPD. None of the playlists belonging to the challenge set have been taken into account in order to avoid overfitting. Our fake set evaluation gives a 30% of success to our recommendations by titles.

Chapter 9

Results

Considering the nature of this work, which is aimed to participate in the RecSys Challenge 2018, it is fairly easy to evaluate the results of the project: by the rating success given by the challenge set. From now on we will focus on describing our challenge set recommendation properties and the results achieved with it.

9.1 Combining the models

We have explained in previous sections how we use our embeddings to predict tracks given tracks, and predict tracks given titles. That is good for playlists that only have title (1000) or for playlists that only have tracks (2000), but what about the other 7000? We should take advantage of both features. After many experiments, and considering the performance of our tracks recommender depending on the number of given seeds (as seen in 7.2) we have decided to use the following percentages.

number of seeds	recommended tracks by tracks model	recommended tracks by titles model
1	100	700
5	200	700
10	300	700
25	400	700
100	600	700

TABLE 9.1: Rrecommending performance for different number of seeds playlists.

Even though we only keep the first 500 tracks, we need more than 500 to start with. After erasing the recommended seed tracks (if any) or the repetitions, we might end up having less than 500.

9.2 Other improvements

It is worth mentioning that we added two additional improvements:

1. Artist titles: Some playlist titles happen to be artist names, so in these cases, before adding any other track recommendation, we recommend to the playlist all the artist tracks.
2. Order: one of the 3 metrics of the Challenge is the number of clicks needed until the first original track shows up, where each click refreshes 10 new tracks. By sorting our tracks recommendation by order of popularity (so by ID), we have been able to escalate many positions in this ranking.

Finally, we can see the recommendations process in the following diagram.

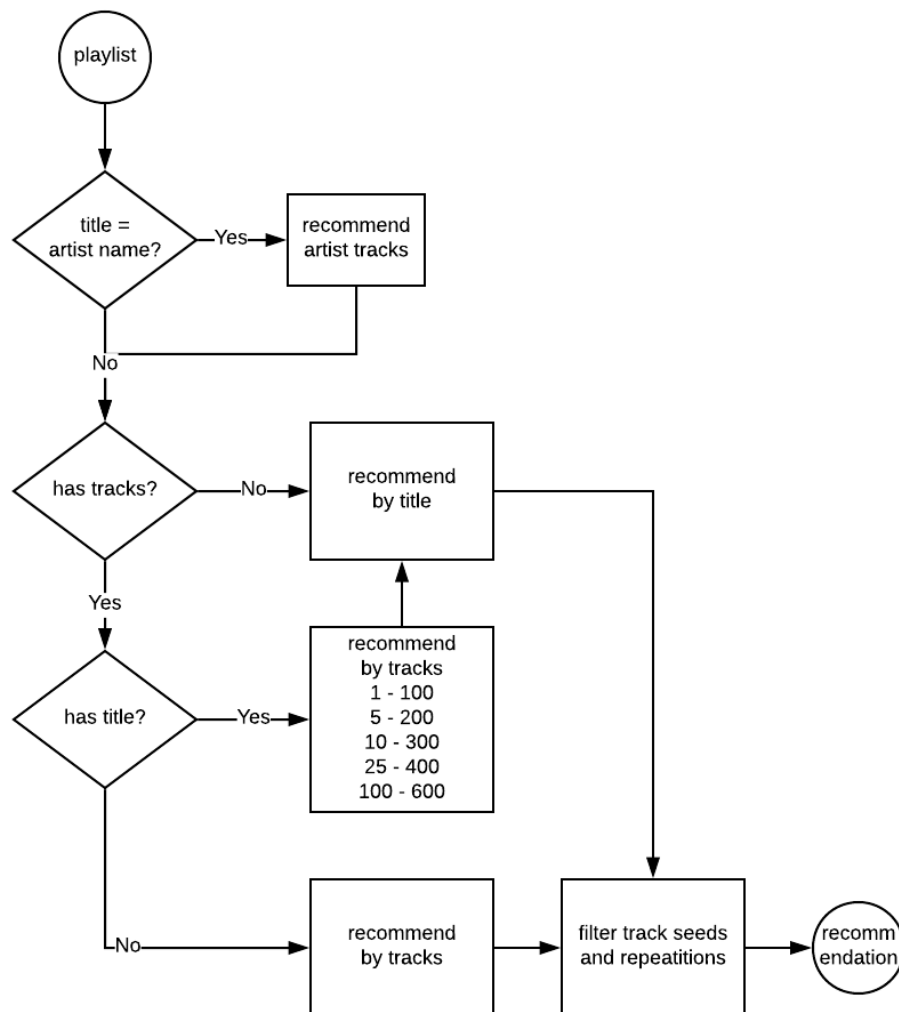


FIGURE 9.1: Tracks recommendation process.

9.3 Ranking

It's time to have a look at our performance during the last two months, since the Recsys Challenge 2018 started. In the following we can see data relative to our performance improvement, submission frequency and relative position on the ranking table.

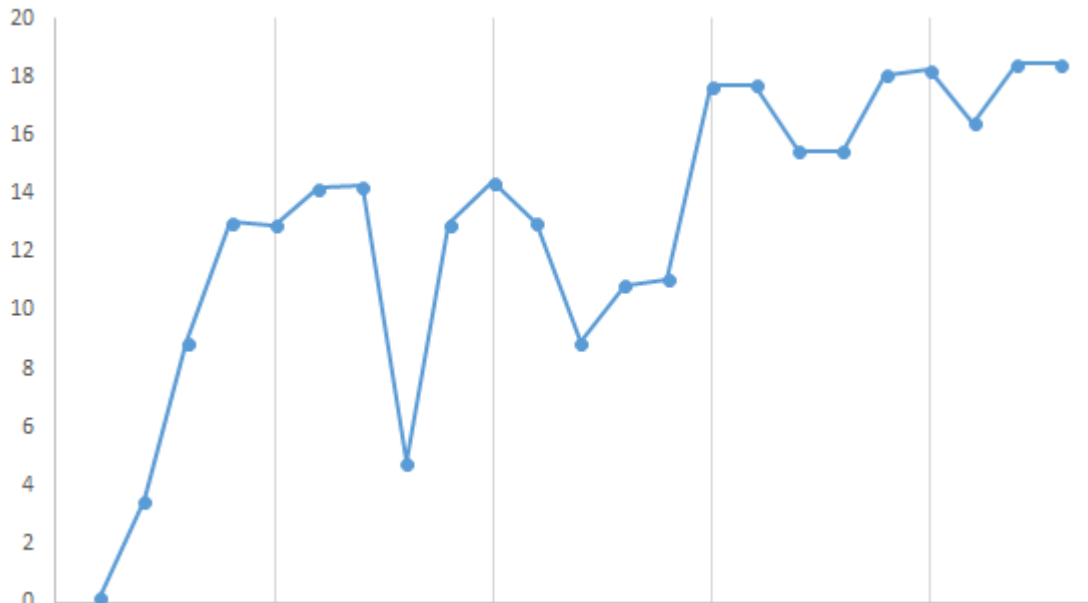


FIGURE 9.2: Precision of recommendations in percentage for all submissions.

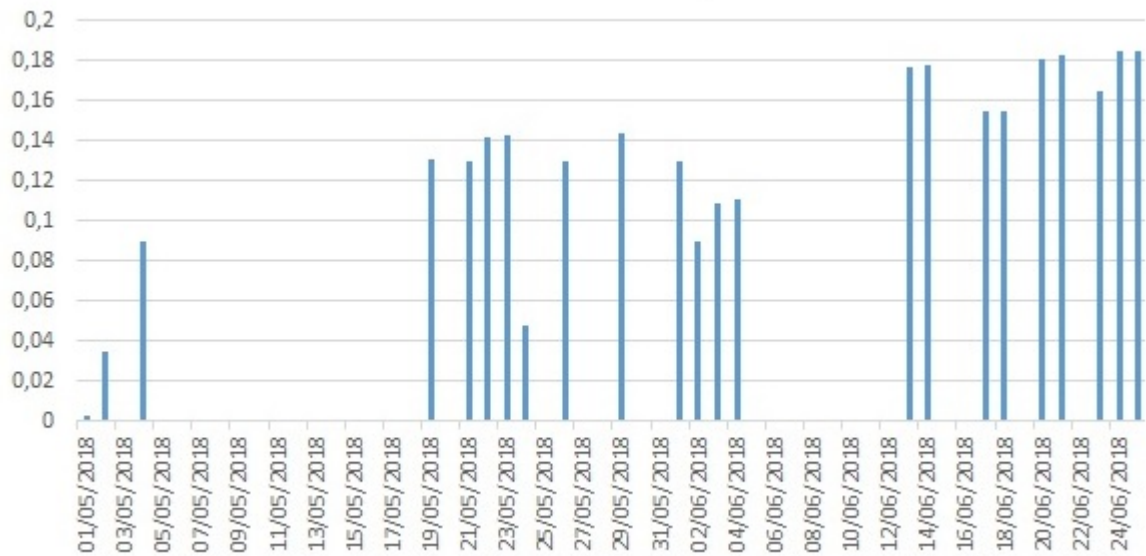


FIGURE 9.3: Submissions by date

As mentioned, there are three rankings depending on three metrics, that sum up to the final classification: Accuracy (9.2), order and number of clicks until the first original track. In figure 9.4 we can see the results of our submissions regarding order vs the number of clicks. They appear to be clearly related. So, by improving the accuracy, the other two metrics improve as well.

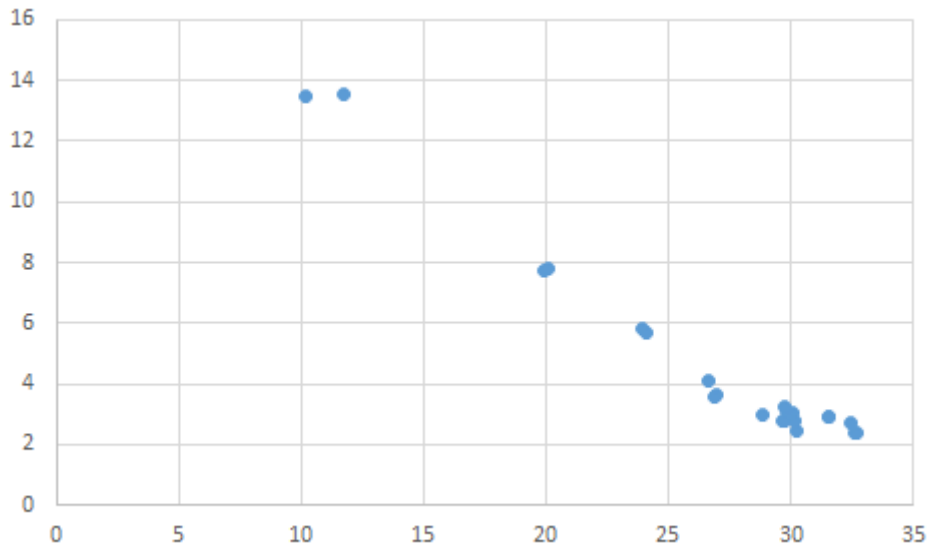


FIGURE 9.4: order vs number of clicks accuracy

Even though we have not stopped improving our recommender over time, so have too the rest of the teams. It must be said that our current 28th position is not final, since the Challenge has not finished yet.

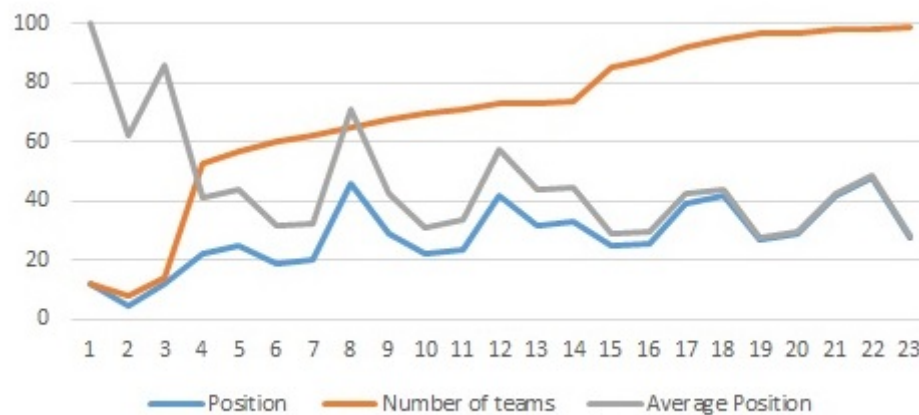


FIGURE 9.5: Number of teams participating over time, and our position in the ranking table intrinsic and relative.

In the appendice can be found a more complete table with the details of each submission.

Chapter 10

Conclusions

Throughout the course of this work, I have learned many different techniques I was hoping to learn and many others that I did not know existed. Understanding the data and getting to work with it was more or less as expected. The state-of-the-art research was also very enriching and I got to learn new concepts such as Markov Chains or n-grams. But the real challenge for me was the practical part.

During a couple months, and with the guidance of my professors, I was trying different approaches and none seemed to work. These failed attempts have not been included in this project, because they have in no way been used for the final recommender. Nevertheless, I believe it is part of the knowledge I acquired and I would like to briefly mention them:

1. Matrix factorization: I created a sparse matrix containing the number of occurrences of two tracks in the same playlist. The matrix was of size 600.000 x 600.000, which is the number of unique tracks appearing more than 5 times in the MPD. It had a sparsity of 99.7%. The matrix was then factorized using SVD reduction down to a 100 dimensions (keeping 87,4% of its variance). Applying the k-MEANS algorithm I hoped to find clusters, but no matter how many clusters I tried to make, one of them always took 80% of the data. The same problem appeared with the playlist-track matrix.
2. Apriori: applying this technique to a generic part of the MPD was either too slow, or produced results only for famous tracks. I then applied it to groups of tracks appearing to same title playlists and I obtained acceptable results, but it was too specific and it had nothing to do with ML.

3. Autoencoder: another possible way to compress the characteristics of our tracks was by means of an autoencoder. However, the dimensions I was working with were too big, and I could not get it to work.
4. Convolutional networks: after applying word2vec to the tracks, I spent some time trying to build a network that would use the track vectors of a playlist to predict another track in the playlist. But word2vec had been poorly trained and the network produced random results.

In summary, I think it is safe to say that I gained a lot of knowledge and I had the chance to get familiarized with some deep learning techniques, which was my main goal in the first place.

I also feel that the frustration I felt at some points during this work are a worth the experience. The fact that problems do not always have a solution is a very logical and well known fact. But it is different to know it than to experience it in your own investigation, in which you have spent many hours and have some expectations.

Lastly, participating in a contest of these characteristics is something I had never done before. Working with a team of experts during these final weeks, meeting, talking, trying new ideas and improving the results has resulted very thrilling and rewarding. It lets me finish this demanding project with a very good feeling.

Appendix A

Appendix

DATE	POSITION	TEAMS	ENCERT	ORDRE	CLICKS	DETAILS
01/05/2018	8	8	0,22	0,86	47.22	embedding mean close tracks
02/05/2018	5	12	3,48	10,07	13.51	500 most popular tracks
04/05/2018	12	14	8,92	19,87	7.80	TITE popular tracks/title, if no title: 500 most popular tracks
19/05/2018	22	53	13,03	26,90	3.70	model dim = 99 , mincount = 10, no shuffle, titles by popularity
21/05/2018	25	57	12,92	26,79	3.64	model dim = 101, min count = 5,no shuffle,10titles by popularity
22/05/2018	19	60	14,21	29,74	3.14	model dim = 100, min count = 5, shuffle, 4titles by popularity
23/05/2018	20	62	14,26	29,96	3.13	model = 100, min= 5, shuffle,by titles similarity \geq 0.7
24/05/2018	46	65	4,80	11,66	13.59	model = 300, min= 5, shuffle,by titles similarity \geq 0.7

TABLE A.1: Features by which some playlists are made, and 5 examples of each.

26/05/2018	29	68	12,97	26,56	4.18	model = 300, min= 5, shuffle, by titles similarity; 0.6 top 60
29/05/2018	22	70	14,41	29,67	3.31	model = 300, min= 5, shuffle, by titles only best title
01/06/2018	24	71	13,01	28,80	3.04	1 track: model=100, else: model=300, by titles only best title
02/06/2018	42	73	8,92	19,98	7.88	TITLE top 100 if similarity; 0.95, no title: 500 most popular tr.
03/06/2018	32	73	10,86	23,82	5.86	model 300, title top 60 if sim; 0.65 PROBLEM:SYMBOLS ERASED MODEL
04/06/2018	33	74	11,08	23,98	5.73	model 300, title top 60 if sim; 0.85 PROBLEM.
13/06/2018	25	85	17,68	31,46	2.96	model 300, title top 20 weighted, if sim; 0.65
14/06/2018	26	88	17,74	31,48	2.95	model 300, title top 1 (if not enough, popular tracks)
17/06/2018	39	92	15,48	29,65	2.85	model 300, title top 10, sim; 0.7 artists, order by pop (if not enough, mess)
18/06/2018	42	95	15,48	29,62	2.86	model 300, title top all sim; 0.95 artists, order by pop (if not enough, mess)
20/06/2018	27	97	18,09	32,35	2.79	model 300, title top 10, sim; 0.95, no artists, filtered by sigmas ; 0.1, (if not enough, popular tracks)
21/06/2018	29	97	18,23	30,19	2.52	model 300, title top 10, sim; 0.95, no artists, filtered by sigmas ; 0.1, (if not enough, popular tracks). For playlist with tracks mix similar songs with songs from similar playlists
23/06/2018	42	98	16,43	30,10	2.84	model 300, title top 10, sim; 0.95, no artists, filtered by sigmas ; 0.1, (if not enough, popular tracks). For playlist with tracks mix similar songs with songs from similar playlists. For playlist with 5 or 10 songs, take songs close the given ones.
24/06/2018	48	98	18,43	32,57	2.47	model 300, title top 10, sim; 0.95, no artists, filtered by sigmas ; 0.1, (if not enough, popular tracks). For playlist with tracks mix similar songs with songs from similar playlists
25/06/2018	28	99	18,46	32,59	2.46	model 300, title top 10, sim; 0.95, artists, filtered by sigmas ; 0.1, (if not enough, top 20). For playlist with tracks mix similar songs with songs from similar playlists

TABLE A.2: Daily submission results to Recsys Challenge 2018. Results and detailed submission model.

Bibliography

- [1] C.-W. Chen Y. Deldjoo-M. Elahi M. Schedl, H. Zamani. *Current Challenges and Visions in Music Recommender Systems Research*. 2018. URL [arXiv:1710.03208v2](https://arxiv.org/abs/1710.03208v2).
- [2] Geoffray Bonnin and Dietmar Jannach. *Automated generation of music playlists: Survey and experiments*. *ACM Comput. Surv.* 47, 2, Article 26 (November 2014), 2014. URL <http://dx.doi.org/10.1145/2652481>.
- [3] Zeno Gantner Lars Schmidt-Thieme Steffen Rendle, Christoph Freudenthaler. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. Machine Learning Lab, University of Hildesheim, 2012. URL [arXiv:1205.2618](https://arxiv.org/abs/1205.2618).
- [4] Robin Burke Negar Hariri, Bamshad Mobasher. *Context-Aware Music Recommendation Based on Latent Topic Sequential Patterns*. *RecSys'12*, September 9–13, 2012, Dublin, Ireland, 2012.
- [5] Beth Logan. *Music recommendation from song sets*. HP Laboratories Cambridge, 2004.
- [6] Malcolm Slaney. *Measuring Playlist Diversity for Recommendation Systems*. Yahoo! Research Labs, 2006.
- [7] Gert Lanckriet Brian McFee. *The natural language of playlists*. 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.
- [8] Douglas Turnbull Thorsten Joachims Shuo Chen, Joshua L. Moore. *Playlist Prediction via Metric Embedding*. *KDD'12*, August 12–16, 2012, Beijing, China, 2012.
- [9] Thorsten Joachims Douglas Turnbull Joshua L. Moore, Shuo Chen. *Learning to embed songs and tags for playlist prediction*. 2012 International Society for Music Information Retrieval (ISMIR), 2012.
- [10] Eduarda Mendes Rodrigues Natasa Milic-Frayling Elena Zheleva, John Guiver. *Statistical models of music-listening sessions in social media*. World Wide Web Conference Committee (IW3C2), 2010.

-
- [11] Claudio BaccigalupoEnric Plaza. *Case-based sequential ordering of songs for playlist recommendation*. T.R. Roth-Berghofer et al. (Eds.): ECCBR 2006, LNAI 4106, pp. 286–300, 2006.
- [12] Arun Swami Rakesh Agrawal, Tomasz Imielinski. *Mining association rules between sets of items in large databases*. ACM SIGMOD Conference Washington DC, USA, 1993.
- [13] Andrew Emery Sylvie Ranwez Michel Crampes, Jean Villerd. *Automatic Playlist Composition in a Dynamic Music Landscape*. SADPI'07, May 21–22, 2007, Montpellier, France, 2007.
- [14] D. Cazaly F. Pachet, P. Roy. *Combinatorial approach to content-based music selection*. IEEE, 2000. URL [10.1109/93.839310](https://doi.org/10.1109/93.839310).
- [15] Ahmed H. Tewfik Masoud Alghoniemy. *User-defined music sequence retrieval*. MULTIMEDIA '00 Proceedings of the eighth ACM international conference on Multimedia, 2000.
- [16] Ahmed H. Tewfik Masoud Alghoniemy. *A network flow model for playlist generation*. IEE, 2003. URL [10.1109/ICME.2001.1237723](https://doi.org/10.1109/ICME.2001.1237723).
- [17] Robin Burke. *Hybrid recommender systems: Survey and experiments*. 2002 Kluwer Academic Publishers, 2001.
- [18] Kai Chen Gregory S. Corrado Tomas Mikolov, Ilya Sutskever and Jeffrey Dean. *Distributed representations of words and phrases and their compositionality*. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, pages 3111–3119, 2013.
- [19] C. McCormick. *Word2Vec Tutorial - The Skip-Gram Model*. <http://www.mccormickml.com>, 2016.
- [20] C. McCormick. *Word2Vec Tutorial Part 2 - Negative Sampling*. <http://www.mccormickml.com>, 2017.
- [21] Greg Corrado Tomas Mikolov, Kai Chen and Jeffrey Dean. *Efficient estimation of word representations in vector space*. CoRR, abs/1301.3781, 2013.
- [22] Yoav Goldberg and Omer Levy. *word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method*. 2014. URL [arXiv:1402.3722](https://arxiv.org/abs/1402.3722).
- [23] Geoffrey Hinton Laurens van der Maaten. *Visualizing Data using t-SNE*. 2008.

List of Figures

3.1	Demographics about Spotify users who originally made the MPD playlists. Age on the left, gender on the right.	4
3.2	MPD percentage left when considering tracks appearing at least n times.	5
3.3	MPD histograms of number tracks, albums, followers and playlists duration in hours.	6
3.4	MPD tracks and playlist titles number of apparitions in the MPD.	6
3.5	Tracks popularity vs average position on the playlists they appear.	7
4.1	Collaborative filtering recommends items to a user based on its similar peers' items.	9
4.2	Audio signal comparison of two songs.	9
5.1	Structure of a biological neuron.	13
5.2	Structure of an artificial neuron.	14
5.3	Activation function examples.	15
5.4	Neural network different structures.	16
5.5	Galton machine.	16
5.6	The plot of cost function vs weight is more or less convex and looks something like this. In black we can see the steps towards the local minima.	17
6.1	Autoencoder schematic functioning.	20
6.2	Word pairs taken from a sentence using a window size of 2. The word highlighted in blue is the input word.	21
6.3	Neural network with one-hot vector input word and one-hot vector output. But when evaluated on an input word, the output vector will actually be a probability distribution (i.e., a bunch of floating point values, not a one-hot vector).	22
6.4	Example of how the output is only affected by the corresponding part of the hidden layer - or a simple matrix multiplication.	23
6.5	Word2vec processing of a word.	23
7.1	Converting playlists to sentences to train our word2vec model.	26
7.2	Word2vec tracks embedding t-SNE representation. 100 tracks per category + 2000 random tracks. Perplexity = 150. Learning rate = 30.	27
7.3	100 tracks per category + 1500 tracks and 3500 tracks respectively. Perplexity=150. Learning rate = 30.	28
7.4	Process of recommending tracks to a playlist using the tracks embedding.	29
9.1	Tracks recommendation process.	37
9.2	Precision of recommendations in percentage for all submissions.	38

9.3	Submissions by date	38
9.4	order vs number of clicks accuracy	39
9.5	Number of teams participating over time, and our position in the ranking table intrinsic and realtive.	39

List of Tables

3.1	MPD stats.	5
7.1	Challenge set playlists features.	28
7.2	Rrecommending performance for different number of seeds playlists. . . .	30
8.1	Features by which some playlists are made, and 5 examples of each. . . .	31
8.2	Playlist titles normalization examples.	32
8.3	9 playlists titles, their ID, and its top 10 most similar playlist titles according to the titles embedding.	34
9.1	Rrecommending performance for different number of seeds playlists. . . .	36
A.1	Features by which some playlists are made, and 5 examples of each. . . .	42
A.2	Daily submission resultsto Recsys Challenge 2018. Results and detailed submission model.	43