Incorporating Word Embeddings in Semantic Label Deep Learning Classifiers

Carlos Ros Perez

A project presented for the MSc Applied Mathematics
of Imperial College London

Year 2020/2021

Abstract

Social tagging, also known as folksonomy, is a common technique for creating datasets based on voluntary cooperation on the part of internet communities. However, these kinds of datasets tend to be affected by the multiplicity of tags describing similar concepts. In this project, we will propose several techniques for maximising the information retrieval from such datasets in the particular case of automatic music classification. These techniques will be based on incorporating word embeddings, which allows us to capture the semantic meaning of the tags, in a deep learning classifier. The methods shall consist in clustering the tags based on the similarity of the word embeddings representing each tag and clustering the tags based on co-occurrence, employing concepts related to word embeddings.

Contents

1 Introduction

In 1962 J. Tukey published the paper "The Future of Data Analysis" [1] in which he drew attention to a new branch of statistics, the statistical analysis of data. He predicted that this new area of knowledge would not only be a branch of mathematics but a science by itself. Almost sixty years later, data science has emerged as an important discipline that combines methods from mathematics, computer science and domain knowledge. The number of papers on data science has increased exponentially over the last decades, increasing by 328% in $2010 - 2019$ compared with $2000 - 2009$ ($N = 244,695$) [2]. This growth has been driven by the increase in computational capacity and the availability of large datasets. In particular, in the last years, general-purpose computing capacity has grown at a rate of approximately 58% per year [3].

Datasets are an essential element of data science Crowdsourcing is a common technique to label information for creating datasets. Crowdsourcing consists of outsourcing tasks, leaving them in charge of a large group of people or an internet community. In this project, we will work with datasets of songs that have been tagged using crowdsourcing. These kinds of datasets usually contain a large number of labelling errors, for this reason, only the most popular tags are used in practice, losing a substantial amount of information from the dataset. The main purpose of this project is to introduce techniques to maximize the retrieval of information from datasets of labelled music without compromising the quality of the information. These methods shall be based on the concept of word embeddings, which is extensively used in natural language processing.

This report is structured as follows. In Section 2 we provide a brief review of the current and past state of automatic music classification. Section 3 consists of an extensive theoretical introduction to the methods used in the experiments; such as neural networks, clustering algorithms or word embeddings. Section 4 and Section 5 provide an overview of the dataset and the computational model respectively. From Section 6 to Section 8 we introduce the different methods for maximizing the information retrieval from the dataset, as well as the results of the numerical experiments. Lastly, in section 9 we propose a different approach for incorporating word embeddings to music tagging that is left for future research.

2 Music tagging and datasets

Music tags refer to any kind of information that can be associated with a song, for instance: genres, emotions, instruments or dates. Automatic music tagging is a relatively new field of study, We may find a few rudimentary examples of music tagging at the end of the last century. For instance, the work of J. A. Foote in 1997, which used spectral methods to distinguish between voice and music employing k-nearest neighbours and random trees as algorithms [4]. One of the first important papers dates from 2002, G. Tzanetakis and P. Cook [5], presented a feature extraction technique combining wavelet transforms, Fourier transforms and filters to compute a Beat histogram, in other words, a representation of the BPM distribution along the track, using such histogram to classify the audio. At that time, most music taggers were based on bags of features models[6, 7, 8, 9, 10]. These models extract features of the tracks, such as the tone or the pitch, and treat them independently to classify the audio. This approach shows a clear drawback, it does not take into account the temporal component of the features or the integration between them, for instance, these models are not sensitive to musical structures such as guitar riffs [11].

The number of models for audio tagging in the scientific literature has increased substantially in the last decade, mainly because of the increasing availability of large datasets of tagged tracks. The state of the art models are mainly based on convolutional neural networks [12, 13, 14, 15, 16], due to their high accuracy for classification tasks. Other works use statistical methods based on Hidden Markov Models [11, 17].

Embeddings are low vector representations of signals; these signals could be, for instance, images, audio or text. Regarding the use of word embeddings for music classification, usually, it is carried out classifying songs according to their lyrics [18, 19, 20]. It is noteworthy to mention a different approach to music classification that incorporates word embeddings, the zero-shot audio classification [21, 22]. Zero-shot learning consists in classifying in classes that were not available during the training set, for instance, we could train a classifier to distinguish rock from classical music and use it to classify jazz songs. It is accomplished using embeddings of audio and text, the classifier is trained on samples (x, y_k), where x is the raw audio and $y_k \in \{y_i\}_{i=1}^n$ is the label associated to x. The training consists in fitting the parameters of the matrix W to verify that $\arg\max_i(\theta(x)^T W \phi(y_i)) = k$ in as many samples (x, y_k) as possible, where $\theta(x)$ and $\phi(y)$ represent audio and word embeddings respectively. We may classify on sets of labels $\{z_i\}_{i=1}^m$ which were not included in the training considering that the predicted label for the audio x is $\arg\max_i(\theta(x)^T W \phi(z_i))$.

Currently, there exist several large datasets publicly available for music tagging, the most popular options are the MagnaTagATune [23], the Million Song Dataset [24] and the MTG-Jamendo Dataset [25]. Usually, only the top 50 tags are used for the classification task [26].

3 Theoretical background

3.1 Learning from data

We may consider that machine learning belongs to the intersection between the fields of data science and artificial intelligence. Machine learning deals with models that are able to make predictions based on a series of parameters that can be tuned in a learning process using training data. There are three main approaches to learning:

- Supervised learning: The model learns from the sample inputs and the expected outputs. It is trained to replicate the outputs of the given training samples.
- Unsupervised learning: The model learns to detect patterns in the input samples without expected outputs, discovering correlations in the inputs.
- Reinforcement learning: The model learns by trial and error. This setting is based on three elements: environment, agent and reward. The environment refers to the inputs in the form of time series, the agent may take actions based on the environment and the reward, which is a function of the environment and the actions taken by the agent. The learning techniques aim to maximize the cumulative reward along the timesteps of the environment.

There exist other kinds of learning that are derived from the foregoing approaches, such as semi-supervised learning [27] or self-supervised learning [28]. Each learning approach is suitable for different problems. In this project we will use supervised and unsupervised learning models. The supervised models shall be feedforward neural networks and convolutional neural networks, introduced in Section 3.2 and Section 3.3 respectively, and the unsupervised learning will be carried out using clustering algorithms presented in Section 3.5.

3.2 Feedforward Neural networks

3.2.1 Introduction

Despite the fact that Neural Networks might seem a cutting-edge technology, their origins date back to the Second World War. In 1943 W. S. McCulloch and W. Pitts described the first computational model based on biological neurons [29]. This system is depicted in Figure (1a), it consist of a function with input $\{x_i\}_{i=1}^{N} \in \{0,1\}^N$ and output:

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{N} x_i > \theta \\ 0 & \text{Otherwise} \end{cases}$$

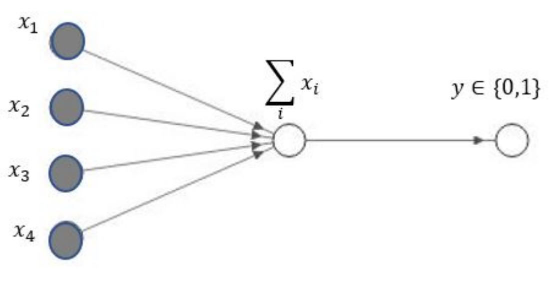
This function is useful as a logic unit; however, it lacks of the fundamental characteristic of the AI, its ability to learn.

The first breakthrough on neural networks occurred in 1958, the year in which psychologist F. Rosenblatt described the perceptron. The perceptron is formed by a series of inputs $\{x_i\}_{i=1}^{N}$, not necessarily binary, combined with a bias term b, that are transformed according to an affine function $f(\mathbf{x}, \mathbf{w}, b) = \sum_{i=1}^{N} x_i w_i + b$ and whose output is the result of applying an activation function $\sigma(h)$ on $f(\mathbf{x}, b)$. In the case of single perceptrons the activation function is the Heaviside function. A single perceptron is suitable for the classification of linearly separable classes. A diagram of the system can be seen in Figure (1b).

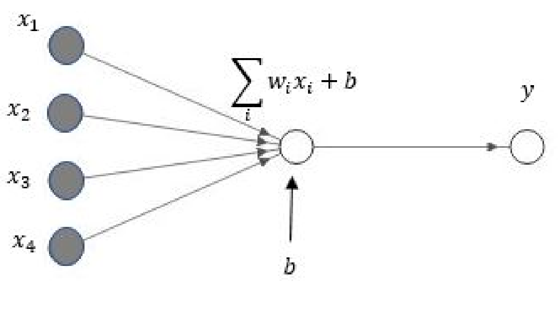
The simplest type of neural networks is the feedforward neural network, also known as multi-layer perceptron, hereinafter FNN, which is defined as the connection of perceptrons. An example of the architecture of FNN is shown in Figure (2). As we can see, this system is structured in layers, the inputs are considered as neurons forming the input layer and the outputs constitute the output layer, the rest of the layers are named hidden layers. We may consider that the network is formed by d hidden layers with $n_i \in \mathbb{N}$ neurons in the i-th layer, where the input and output layers are the layers 0 and $d+1$ respectively. Let $\sigma_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ be the activation function of the layer $i > 0$, in general, the activation function of the multilayer perceptron is not the Heaviside function. Let $\boldsymbol{f}^i(\boldsymbol{x}) = \boldsymbol{x}W^{i-1} + \boldsymbol{b}^{i-1}$, where $\boldsymbol{x} \in \mathbb{R}^{n_{i-1}}$, $W^{i-1} \in \mathbb{R}^{n_{i-1} \times n_i}$ and $\boldsymbol{b}^{i-1} \in \mathbb{R}^{n_i}$, where all the vectors are row vectors. We may define the FNN as the function $\boldsymbol{g}_\theta$:

$$\boldsymbol{g}_\theta = \sigma_{d+1} \circ \boldsymbol{f}^{d+1} \circ \cdots \circ \sigma_1 \circ \boldsymbol{f}^1 \tag{1}$$

Where θ represents the set of parameters $\{W^i, \mathbf{b}^i\}_{i=1}^{d+1}$ implicitly contained in the definition of



(a) McCulloch Pitts neuron



(b) Perceptron

Figure 1: Basic neurons

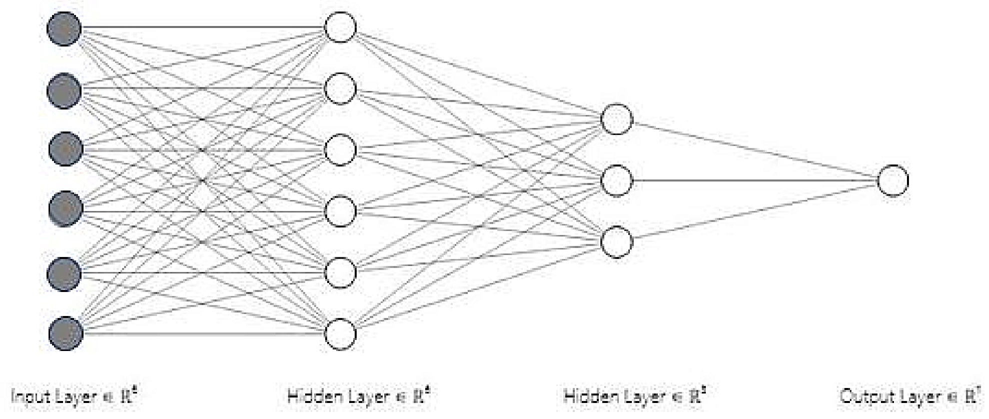


Figure 2: Neural network as the connection of perceptrons

$\boldsymbol{g_\theta}$. From Equation (1) the output of the i-th layer, which we shall denote as $\boldsymbol{h}_i \in \mathbb{R}^{n_i}$ for $i > 0$, is given by the recursive formula:

$$\boldsymbol{h}_i = \sigma_i(\boldsymbol{h_{i-1}} W^{i-1} + \boldsymbol{b}^{i-1}) \tag{2}$$

Being $\boldsymbol{h}_0$ the vector of input variables. Again, all vectors are row vectors.

It can be shown that the FNN $\boldsymbol{g_\theta}$ may approximate any function $f \in C(\mathcal{X}, \mathbb{R}^{n_{d+1}})$, where $\mathcal{X}$ is a compact subset of $\mathbb{R}^{n_0}$ [30]. The Universal Approximation Theorem states that $\forall \varepsilon$ and $\forall f \in C(\mathcal{X}, \mathbb{R}^{n_{d+1}})$ there exist a FNN $\boldsymbol{g_\theta}$ such that:

$$\sup_{x \in \mathcal{X}} \|\boldsymbol{g_\theta} - f(x)\| < \varepsilon$$

In other words, the set of FNN is dense in $C(\mathcal{X}, \mathbb{R}^{n_{d+1}})$.

3.2.2 Activation functions

There is no general rule for defining activation functions, except that they tend to be increasing and non-linear, except for the identity activation. In the following table, we show a non-exhaustive list of common activation functions [31], see that they are applied to inputs in $\mathbb{R}^{n_i}$ element-wise:

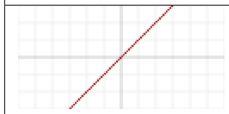
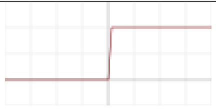
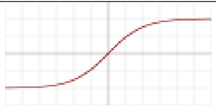
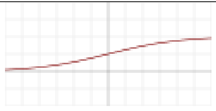
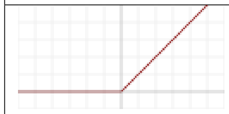
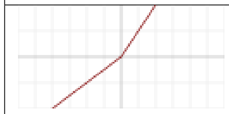
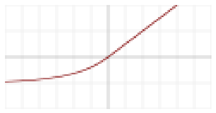
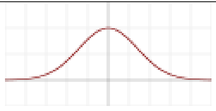
Graph	Name	function $f(x)$	Derivative $f'(x)$	Range
	Linear	x	1	$(-\infty, \infty)$
	Heaviside	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
	Hyperbolic tangent	$\tanh(x)$	$1 - f(x)^2$	$(-1, 1)$
	Logistic	$\frac{1}{1+e^{-x}}$	$f(x)(1 - f(x))$	$(0, 1)$
	Rectified Linear Unit (ReLU)	$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
	Parameteric Rectified Linear Unit (PReLU)	$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$
	Exponential linear unit (ELU)	$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \\ \text{undefined} & \text{if } x = 0 \text{ and } \alpha \neq 1 \end{cases}$	$(-\infty, \infty)$
	Gaussian	e^{-x^2}	$-2xe^{-x^2}$	$(0, 1]$

Table 1: Activation functions

Some decades ago, the sigmoid and *tanh* were the most popular activation functions; however, due to its computational simplicity and high performance in training, the ReLU is the most used activation function nowadays [32]. The drawback of the ReLU is its lack of activation when all the inputs are negative, which is known in the literature as the dead ReLU problem, nevertheless, this characteristic is not a problem in most cases and can be solved using a PReLU or an ELU function [33].

There exist another kind of activation function, commonly used for multi-class classification, known as multi-dimensional activation functions. These activation functions act on the output of all the neurons of the output layer. The most important ones are summarized in Table (2).

In the computational models we will use the ReLU activation function in the hidden layers, the sigmoid activation in the output layers and the softmax activation for creating embeddings.

Name	Domain and codomain	Function $f_i(\mathbf{x})$	Derivative $\frac{\partial f_i(\mathbf{x})}{\partial x_j}$	Range
Softmax	$\mathbb{R}^n \to \mathbb{R}^n$	$\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$	$f_i(\mathbf{x})\left(\delta_{ij} - f_j(\mathbf{x})\right)$	$(0,1)$
Maxout	$\mathbb{R}^n \to \mathbb{R}$	$\max_i x_i$	$\begin{cases} 1 & \text{if } j = \underset{i}{\operatorname{argmax}} x_i \\ 0 & \text{if } j \neq \underset{i}{\operatorname{argmax}} x_i \end{cases}$	$(-\infty,\infty)$

Table 2: Multi-dimensional activation functions

3.2.3 Loss functions

First of all, we need to establish an objective measure of the performance of the FNN. This measure is the so-called loss function. In the case of supervised learning in a FNN with n_{d+1} output neurons, let $\hat{\mathbf{y}} \in \mathbb{R}^{n_{d+1}}$ be the optimal prediction for the input $\mathbf{x} \in \mathbb{R}^{n_0}$, the per-example loss function is a function $\ell : \mathbb{R}^{n_{d+1}} \times \mathbb{R}^{n_{d+1}} \to \mathbb{R}$ such that its global minimum is located in $\hat{\mathbf{y}}$ [34].

When training a FNN, we do not use one example at each iteration, instead, we pass the whole dataset or, more commonly, a batch of the data. Let $\mathcal{M}$ be a set of samples $(\mathbf{x}_i, \mathbf{y}_i)$, the loss function is defined as:

$$\mathcal{L}(\theta; \mathcal{M}) := \frac{1}{|\mathcal{M}|} \sum_{\mathbf{x}_i, \mathbf{y}_i \in \mathcal{M}} \ell\left(\mathbf{y}_i, \mathbf{g}_\theta\left(\mathbf{x}_i\right)\right) \tag{3}$$

Where $\mathbf{g}_\theta$ is the function that represents the FNN defined in Equation (1). In the case of regression a common choice of the loss function is the mean squared error (MSE):

$$\mathcal{L}(\theta; \mathcal{M}) = \frac{1}{|\mathcal{M}|} \sum_{\mathbf{x}_i, \mathbf{y}_i \in \mathcal{M}} \left(\mathbf{y}_i - \mathbf{g}_\theta\left(\mathbf{x}_i\right)\right)^2 \tag{4}$$

In the case of discrete outputs, the FNN is a classifier. If we classify inputs into n classes, we shall use an output layer with n neurons, where each neuron correspond to a different class. We may divide the classification problems into three categories:

- Binary classification: Classifying inputs into two exclusive classes
- Multi-class classification: Classifying inputs into more than two exclusive classes.
- Multi-label classification: Classifying inputs into $n \geq 2$ non-exclusive classes.

In the computational models we will use multi-label classifiers. Let $t \in \{0,1\}^n$ be the ground truth of the sample, i.e., $t_i = 1$ if class i is one the true labels of the sample and $t_i = 0$ otherwise. In general, the loss function for a classifier is based on the cross-entropy, defined as:

$$\mathcal{L}\left(\theta; \mathcal{M}\right) = -\frac{1}{n}\sum_{i}^{n} t_i \log\left(\hat{y}_i\right) \tag{5}$$

Where $\hat{y}_i \in (0,1)$ is the predicted probability for class i, see that the predictions are continuous variables. The cross-entropy, as stated in Equation (5), is suitable for multi-class classification. For binary classification, i.e., $n = 2$ exclusive classes, Equation (5) is reduced to the so-called binary cross-entropy:

$$\mathcal{L}\left(\theta; \mathcal{M}\right) = -\frac{1}{2}\sum_{i}^{2} t_i \log\left(\hat{y}_i\right) = -\frac{1}{2}\left(t_1 \log\left(\hat{y}_1\right) + \left(1 - t_1\right)\log\left(1 - \hat{y}_1\right)\right) \tag{6}$$

In the case of multi-label classification, which is the one that we use in the computational models, we may consider the classification into n non-exclusive classes as n different binary classifications. The FNN would apply a sigmoid function to the output, as shown in Table (1), to map the result into $(0,1)$, therefore, $\hat{y}_i$ would correspond to the probability that the input belongs to class i. Under this premise, we may reuse the expression of the binary cross-entropy, Equation (6), adjusting the normalization we would get:

$$\mathcal{L}\left(\theta; \mathcal{M}\right)\left(\hat{y}_i\right) = -\frac{1}{n}\sum_{i=1}^{n}\left(t_i \log\left(\hat{y}_i\right) + \left(1 - t_i\right)\log\left(1 - \hat{y}_i\right)\right) \tag{7}$$

See that cross-entropy is more suitable for classification than MSE because it penalises the classifier more severely, since $(t_i - y_i)^2 \in (0,1)$ whilst $-t_i \log(y_i) \in (0, \infty)$.

3.2.4 Gradient descent and optimisers

The minimum of the loss function corresponds with the optimal predictions, in other words, to the optimal choice of parameters θ. Therefore, training the FNN is reduced to the optimisation problem consisting in finding the parameters θ that minimise the loss function. It is not surprising that, in general, there is no analytical solution for this problem. Thus, we must use numerical methods based on gradient methods, since the negative gradient points toward the direction of the steepest descent of the function. The basic gradient method is the Gradient Descent method, which is defined as follows:

$$\theta_{n+1} = \theta_n - \eta_n \nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}\right) \tag{8}$$

Where θ_n is the values of the parameters at the n-th iteration and η_n is the learning rate, a number that can be used to tune the extent of variation of the parameter between iterations. On the one hand, if the learning rate is too small, it could lead to a slow convergence speed and the possibility of iterate around a local minimum without reaching the global minimum. On the other hand, a large learning rate can lead to instabilities in the iterations, which will prevent convergence to occur because the weights jump around the minimum without reaching it. The optimal value of the learning rate depends on the problem, usually, it is set around $\eta = 0.01$; however, for problems with a large amount of data it is convenient to choose learning rates that are small and constant or varying in size [35].

The gradient descent expressed in Equation (8) is never used in practical application due to a major drawback of the formula. For large training sets $\mathcal{M}$ we would have to pass all the samples of the dataset per iteration, leading to a sluggish convergence speed for small learning rates or an inaccurate convergence for large learning rates. The simplest solution is updating the parameters more frequently. For this purpose, we may divide the training set $\mathcal{M}$ into subsets of training data ($\mathcal{M}_m$), called batches, using a batch per iteration instead of the whole dataset $\mathcal{M}$. This leads to the Stochastic Gradient Descent (SGD) method[1]:

$$\theta_{n+1} = \theta_n - \eta_n \nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}_m\right) \tag{9}$$

We use the term epoch to refer to the fact of iterating a training algorithm on all the samples of a dataset ($\mathcal{M}$) using batches of data ($\mathcal{M}_m$).

A simple way to accelerate the convergence is adding momentum to the parameter update, the SGD with momentum is defined as follows [36]:

$$\theta_{n+1} = \theta_n - \eta_n \nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}_m\right) + \mu_n\left(\theta_n - \theta_{n-1}\right) \tag{10}$$

The parameter μ_n is the momentum term and verifies $0 \leq \mu_n \leq 1$. Usually, it is set at $\mu_n = \mu = 0.9$; however, selecting the optimal values of μ_n and η_n usually is a matter of trial and error.

A common characteristic in the GD algorithm is that they use the same step size in all the components of θ. For instance, let us consider that $\mathcal{L}\left(\theta_n; \mathcal{M}_m\right)$ is a maximum with respect to the component θ_a but it is a minimum with respect to θ_b, in that case, it would be convenient to increase the learning rate in the direction of θ_a and decrease it in the direction of θ_b. This is the

[1]Some authors refer to this method as the Mini-Batch Gradient Descent

idea that led to the Adagrad algorithm, which can be stated element-wise as [37]:

$$\begin{aligned} \theta_{n+1}[i] &= \theta_n[i] - \eta \frac{\nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}_m\right)[i]}{\sqrt{s_n[i]} + \varepsilon} \\ s_n[i] &= s_{n-1}[i] + \left(\nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}_m\right)[i]\right)^2 \end{aligned} \tag{11}$$

Where $s_0 = \nabla_\theta \mathcal{L}\left(\theta_0; \mathcal{M}_m\right)^2$ and $\mathbf{x}[i]$ represents the i-th component of $\mathbf{x}$. One of the most common optimisers in practical applications is the Adam optimizer. It combines the above ideas creating an optimiser with adaptive learning rates and momentum. It is defined as [38]:

$$\begin{aligned} \theta_{n+1} &= \theta_n - \eta \frac{\hat{m}_n}{\sqrt{\hat{v}_n} + \varepsilon} \\ \hat{m}_n &= \mathbb{E}[g_n] / \left(1 - \beta_1\right) \\ \hat{v}_n &= \mathbb{E}\left[g_n^2\right] / \left(1 - \beta_2\right) \\ \mathbb{E}[g_n] &= \beta_1 \mathbb{E}[g_{n-1}] + \left(1 - \beta_1\right) \nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}_m\right) \\ \mathbb{E}\left[g_n^2\right] &= \beta_2 \mathbb{E}\left[g_{n-1}^2\right] + \left(1 - \beta_2\right)\left(\nabla_\theta \mathcal{L}\left(\theta_n; \mathcal{M}_m\right)\right)^2 \end{aligned} \tag{12}$$

Where $g_0 = \nabla_\theta \mathcal{L}\left(\theta_0; \mathcal{M}_m\right)$. The Adam optimiser is computationally efficient, uses little memory and is invariant to rescaling of the gradients [38]. The last property is desirable in cases where the magnitude of the gradient is small. We will employ the Adam optimizer in the computational model.

3.2.5 Backprogation

As we have seen, optimising the parameters of the FNN is a simple task based on two steps:

1. Computing the gradient with respect to the parameters of the lost function evaluated at $\mathcal{M}_m$.
2. Updating the parameters using the gradient.

We recall that the parameters of the FNN are $\theta = \{W^i, \mathbf{b}^i\}_{i=1}^{d+1}$ and the recursive formula for computing the output of the i-th layer is expressed in Equation (2). The loss function is a function of the sample outputs ($\mathbf{y}$) and the predicted outputs ($\hat{\mathbf{y}}$). Where $\hat{\mathbf{y}} = \sigma_{d+1}(\boldsymbol{h_d} W^d + \boldsymbol{b}^d)$. Based on Equation (2), we may use function composition to express $\hat{\mathbf{y}}$ as a function of $\mathbf{h}_k$ and $(W^{k'}, \boldsymbol{b}^{k'})$ with $k' \in \{k, \cdots, d\}$. Since $\mathcal{L}\left[\theta; \mathcal{M}_m\right](\mathbf{y}, \hat{\mathbf{y}})$ can be expressed in terms of $(\mathbf{y}, \mathbf{h}_k, W^k, \mathbf{b}^k)$ using function composition, we may compute the derivatives of $\mathcal{L}$ with respect to W_{ij}^k and b_i^k employing the chain rule. Recalling that $\boldsymbol{f}^i(\boldsymbol{h}_{i-1}) = \boldsymbol{h}_{i-1}W^{i-1} + \boldsymbol{b}^{i-1}$, we would arrive at the following expressions:

$$\delta_p^k = \frac{\partial \mathcal{L}}{\partial f_p^k} = \sum_{j=1}^{n_{k+1}} \frac{\partial \mathcal{L}}{\partial f_j^{k+1}} \frac{\partial f_j^{k+1}}{\partial f_p^k} = \sum_{j=1}^{n_{k+1}} \delta_p^{k+1} \frac{\partial f_j^{k+1}}{\partial f_p^k} = \sigma'\left((\mathbf{h}_k)_p\right) \sum_{j=1}^{n_{k+1}} W_{pj}^k \delta_j^{k+1}$$

$$\frac{\partial \mathcal{L}}{\partial W_{qp}^k} = \frac{\partial L_i}{\partial f_p^{k+1}} \frac{\partial f_p^{k+1}}{\partial W_{qp}^k} = h_q^k \delta_p^{k+1}$$

$$\frac{\partial \mathcal{L}}{\partial b_p^k} = \delta_p^{k+1}$$

Given that we know the value of the parameters and the inputs $\mathbf{x}$, we may compute $\boldsymbol{f}^i$ for each layer i. The backpropagation algorithm consist in computing $\boldsymbol{\delta^{d+1}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{f}^{d+1}}$ and using it, along with $\boldsymbol{f}^i$, to obtain the values of $\boldsymbol{\delta^i}$. Finally, we may apply the above formulas to deduce the value of the gradients $\frac{\partial \mathcal{L}}{\partial W_{qp}^k}$ and $\frac{\partial \mathcal{L}}{\partial b_p^i}$

3.2.6 Preventing overfitting

For training a FNN, the dataset must be split into a training set and a validation set. Otherwise, we would face a common problem of FNN called overfitting. It consists in adjusting the weight exclusively to reduce the loss function on the training set, losing the ability to generalize correctly. See that the loss function is not only a function of the parameters (θ), but also of a set of samples ($\mathcal{M}_m$). We perform the training on the training set, the unique purpose of the validation set ($\mathcal{M}_v$) is to evaluate the loss function $\mathcal{L}(\theta; \mathcal{M}_v)$ after each epoch. We recall that an epoch is the number of iterations necessary to pass all the training data in form of batches to the training algorithm.

We may prevent overfitting using regularisation. Below we show three common regularisation techniques:

- ℓ^2 and ℓ^1 regularisation: This technique consist in adding an extra term in the loss function with the form $\mathcal{L}'(\theta; \mathcal{M}_m) = \lambda\|\theta\|_2^2 + \mathcal{L}(\theta; \mathcal{M}_m)$ for ℓ^2 or $\mathcal{L}'(\theta; \mathcal{M}_m) = \lambda\|\theta\|_1 + \mathcal{L}(\theta; \mathcal{M}_m)$ for ℓ^1. Where $\lambda \in \mathbb{R}$ is an hyper-parameter, i.e., a parameter that determines the architecture of the FNN and cannot be tuned using gradient methods. This extra parameter prevents overfitting because penalises large values of the weight and the bias

- Dropout: This technique is applied over certain layers. During each iteration of the training, each neuron of such layers is activated with probability p, usually p = 0.5, or is canceled, in other words, its output is changed to zero, with probability 1 - p. This technique prevents an excessive adaptation of the neurons to the training data [39].

- Early stopping: This is the simplest regularisation technique, it consists in stopping the training if the loss function on the validation set does not decrease after n epochs, where n is a hyper-parameter.

We shall use the dropout regularization technique to prevent overfitting in the computational model.

3.2.7 Batch normalization

Batch normalization, introduced by S. Ioffle and C. Szegedy in 2015 [40], is a technique that reparametrizes the inputs of a neural network to facilitate the training. See that the input of any layer is the output of the previous layer, therefore, the change of the weight of the layers during training may change the distribution of the inputs of the following layers $\boldsymbol{h}_i$, this phenomenon is known as internal covariate shift. It may slow down the training since it would require reducing the learning rates to achieve convergence [40]. Batch normalization avoids the internal covariate shift standardizing the input of each later of the network along the batch. Let us assume that we train the network using batches of samples of the form $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^m$, we will denote as $\{\boldsymbol{h}_i\}_{i=1}^m$ the input of an arbitrary layer in the network produced by the foregoing batch of samples. The batch normalization consists in substituting the input $\{\boldsymbol{h}_i\}_{i=1}^m$ for $\{\gamma\tilde{\boldsymbol{h}}_i + \beta\}_{i=1}^m$ in the training process, where $\{\tilde{\boldsymbol{h}}_i\}_{i=1}^m$ is given by:

$$\begin{aligned}
\boldsymbol{\mu} &= \frac{1}{m}\sum_{i=1}^{m} \boldsymbol{h}_i \\
\sigma^2 &= \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{h}_i - \boldsymbol{\mu})^2 \\
\tilde{\boldsymbol{h}}_i &= \frac{\boldsymbol{h_i} - \boldsymbol{\mu}}{\sqrt{\sigma^2 + \varepsilon}}
\end{aligned}$$

Where ε is a small parameter, usually 10^{-8}, which is included for numerical stability. The hyper-parameters γ and β from the linear combination $\gamma\tilde{\boldsymbol{h}}_i + \beta$ are trained using backpropagation as any other parameter.

3.3 Convolutional Neural Networks

3.3.1 Introduction

Apart from the feedforward neural networks, there exist a whole range of neural networks aimed to solve different problems. In this section, we are going to focus on a kind of network that we will use extensively in this project, the convolutional neural network. Its origin, once again, comes from the analogy between the human brain and the computational systems. It is based on the research of the visual cortex carried out by D. Hubel and T. Wiesel [41], they proposed that the human ability to recognise patterns was based on the interaction of two kinds of cells in the visual cortex, one of them would recognise features and the other would generalise this features by means of deformations. This model inspired K. Fukushimsa [42] to create the neocognitron, which included two kinds of layers, the convolutional layers and the downsampling layers. This model was improved by Y. LeCun [43] implementing the backpropagation algorithm and giving raise to the modern convolutional neural networks (CNN).

We may define the CNN as multilayer networks in which some layers perform a convolution operation instead of matrix multiplication of inputs by weights. In CNN we must not understand the layers as a set of neurons but as a unique filer. This filter could act on inputs with more than one dimension. There are three kinds of layers in a CNN: convolutional layers, pooling layers and dense layers. The dense layers are made of fully connected perceptrons, they are equivalent to the hidden layers of the FNN, as can be seen in Figure (2). In CNN, these kinds of layers tend to be located at the end of the network.

3.3.2 Convolutional Layers

The convolution is an integral transform performed on two functions to produce a third one that is usually considered as a filtered version of one of the input functions. It is defined in the following way:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau \tag{13}$$

In this case we are interested in the discrete version of the convolution between an input $x[n]$ and a filter $h[n]$. In addition, the domain of the filter is finite, in other words, it vanishes except if $n \in \{0, \cdots, K-1\}$. We may define the discrete convolution filter as:

$$(x \star h)[n] = \sum_{k=0}^{K-1} h[k]x[n-k] \tag{14}$$

In most cases, the input $x[n]$ is two-dimensional, such as images for computer vision or spectrograms for audio processing. We may define he convolution between a signal $x[n_1, n_2]$ and a filter $h[n_1, n_2]$ with shape $K_1 \times K_2$ as:

$$(x \star h)[n_1, n_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} h[k_1, k_2]\, x[n_1 - k_1, n_2 - k_2] \tag{15}$$

Usually, the CNN use a slight modification of the convolution called cross-correlation, defined as:

$$(x \star h)[n_1, n_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} h[k_1, k_2]\, x[n_1 + k_1, n_2 + k_2] \tag{16}$$

The convolutional layers take an input $x[n_1, n_2]$ ($x[n]$) and performs convolution between the input and a filter $h[n_1, n_2]$ ($h[n]$), then, it applies a bias and an activation function. Formally, if $\boldsymbol{h}_i$ is the input of the convolutional layer with filters $\{\mathbf{k}_i^{(n)}\}_n$, the output is given by:

$$\mathbf{h}_{i+1}^{(n)} = \sigma\left(\left(\mathbf{h}_i^{(n)} * \mathbf{k}_i\right) + b_i\right) \tag{17}$$

It is noteworthy to discuss briefly the dimensionality in convolutional layers. Each layer can apply various filters, which are implicit in the dimensionality of the layer.

Let us see an example with an image. Let us assume that the convolutional layer passes 2D filters, the dimensions of an array representing an image formed by $n \times m$ pixels are $n \times m \times 3$, see that the image is stored as three different $n \times m$ arrays, each one encoding just tonalities of one of the basic RGB colours. If a convolutional layer passes ℓ 2D filters, it would be the same as passing a filter $\mathbf{k} \in \mathbb{R}^{x \times y \times 3 \times \ell}$. The output would have dimension $u \times v \times \ell$, where $u = n - x + 1$ and $v = m - y + 1$ are called spatial dimensions and ℓ is called depth dimension or number of channels. The channels can be added to the cross-correlation trivially, using the notation from the previous example:

$$(x \star h)[n_1, n_2, b] = \sum_{k_1=0}^{x-1} \sum_{k_2=0}^{y-1} \sum_{a=0}^{2} h[k_1, k_2, a, b]\, x[n_1 + k_1, n_2 + k_2, a] \tag{18}$$

Let us assume a spatial axis where the length of the input and the filters are a and b respectively, then, from Equation (17), the length of such axis in the output is $d = a - b + 1$.

We could modify the size of the output in a particular axis using padding or strides. Padding consists in adding an array of zeros with length p to a given axis in the input. In this case, the output in the foregoing axis would have length $d' = a + p - b + 1$ after passing the convolutional layer. The technique of strides consists in excluding certain outputs of cross-correlation, based on a parameter[2] $s \in \mathbb{N}$. Recalling the formula of the cross-correlation (17), we would exclude all the values of $(x \star h)[n_1, n_2]$ except the ones that verify $(n_1, n_2) = (ms, ls)$ with $(m, l) \in \mathbb{N}_0^2$ and reassigning the values $(x \star h)[n_1, n_2] \leftarrow (x \star h)[sn_1, sn_2]$. In such manner, the length of all the spatial dimensions would be reduced, the dimension of the input in the dimension considered previously would be $d' = \left\lfloor \frac{a-b}{s} \right\rfloor + 1$.

3.3.3 Pooling Layers

Clustering layers are an important component of CNN whose goal is to reduce the size of the input and provide invariance in small translations of the input. In most cases, pooling layers are applied on spatial dimensions, but we could apply pooling layers on the channels. Let us assume that the input has size $n \times m$, first, the pooling layer splits the input into disjoint subarrays with dimension $n' \times m'$, such that n' and m' divide exactly to n and m respectively, then, it applies a function that maps the subarrays into $\mathbb{R}$. The two main kinds of pooling layer are the max pooling layer and the average pooling layer.

- The max pooling layer applies the max function over the components of the subarrays. For instance, using $(m, n) = (4, 4)$ and $(m', n') = (2, 2)$:

$$\begin{bmatrix} 6 & 2 & 15 & 3 \\ 5 & 14 & 8 & 11 \\ 8 & 9 & 12 & 7 \\ 11 & 3 & 4 & 13 \end{bmatrix} \Rightarrow \begin{bmatrix} 14 & 15 \\ 11 & 13 \end{bmatrix}$$

- The average pooling layer computes the average of the elements of the subarrays. For instance, applying it on the above matrix:

$$\begin{bmatrix} 6 & 2 & 15 & 3 \\ 5 & 13 & 8 & 11 \\ 8 & 9 & 12 & 7 \\ 11 & 3 & 4 & 13 \end{bmatrix} \Rightarrow \begin{bmatrix} 6.5 & 9.25 \\ 7.75 & 9 \end{bmatrix}$$

The most common pooling layer is the max pooling, since it tends to improve the performance of the CNN compared with the average pooling [44].

In a CNN, pooling layers are often placed between convolutional layers. There are two main motivations for using them:

[2]$\mathbb{N}$ indicates the set of natural numbers without 0 and $\mathbb{N}_0$ is the set of natural numbers including 0

- They reduce the dimensions and also the number of small values in the input, which usually do not carry any information. This leads to an improvement in the computational capacity since the CNN has to process fewer components.

- They provide an approximate invariance under small translations in the input, since the functions applied by the pooling layers just depend on the values of the subarrays.

For completeness, In Figure (3) we have depicted an example of the architecture of a CNN

3.3.4 Comparing CNN and FNN

We have summarized the characteristics of CNN, these attributes make them suitable for some problems such as computer vision or speech detection [45]. For instance, let us consider the problem of recognising numbers in an image. Let us assume that we pass an image of a number one. For the FNN the feature vector would consist of a flattened version of the pixel values of the image. The network would adjust its parameters to increase the chances of recognising a 1 for a feature vector similar to the one given. However, we may feed the FNN with another image of a number one that translates into a different feature vector, leading to inaccuracies. For instance, if the number were set in a different position or the image is taken with different illumination or angle.

On the other hand, the CNN trains its parameters to recognise features. The convolutional layers perceive shapes, therefore they are less sensitive to small changes in the images, due to their approximate invariance under slight translations. In addition, the CNN are less prone to overfitting than FNN, since the former tend to use fewer parameters than the latter [46].

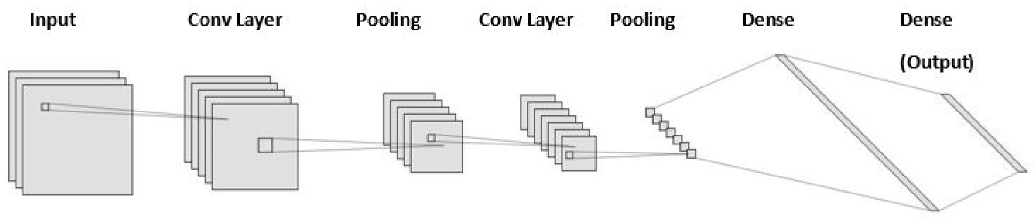


Figure 3: Example of CNN

3.4 Performance metrics

Whatever type of network we decide to use we could take something for granted, there will be errors. That is why assessing the performance of the networks is an important part of the learning process. Regarding regression models, the usual way to assess the performance is using the loss function. If we consider a binary classifier, it could fail in many different ways. For instance, two classifiers could have the same error rates, but one of them could commit more errors in one of the states. Let us consider the binary classifier, in this case, there are only four states of being wrong or right, they can be summarized in a confusion matrix:

Actual class / Predicted	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Table 3: Confusion matrix

Where the acronyms stand for the following terms:

- TP (True Positives): Amount of positive samples that the classifier is able to classify correctly.
- FP (False positives): Quantity of negative samples that the classifier classifies as positive
- FN (False negatives): Number of positive samples wrongly classified as positive
- TN (True negatives): Amount of negative samples correctly classified

From these four quantities, we may define several empirical metrics:

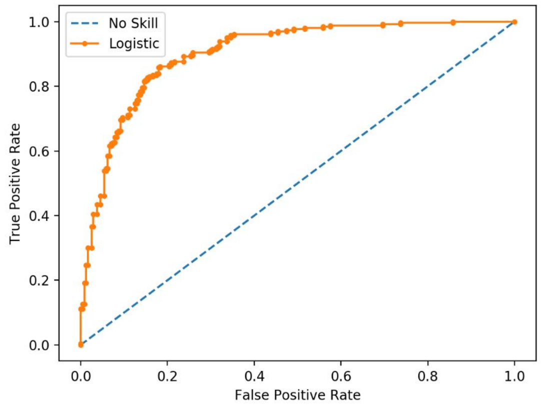
- True positive rate, also known as recall or sensitivity:
$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$
- True negative rate or specifity:
$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}$$
- Precision:
$$\text{precision } = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}$$

The suitability of each metric depends on the problem. There exist other kinds of metrics that are commonly used to assess the performance of classifiers based on the variation of the metrics with respect to the thresholds. These metrics are the AUC ROC and the AUC PR. To understand them, we must recall how does a classifier work. In binary classification, the output layer consists of one neuron. The prediction would be a number $\hat{y} \in (0, 1)$, in order to assign a class to the prediction we must establish a threshold τ such that the input is included in class 0 if $\hat{y} < \tau$ and in class 1 otherwise. The value of the hyper-parameter τ will determine the components of the confusion matrix (3) and, accordingly, the value of the empirical metrics.

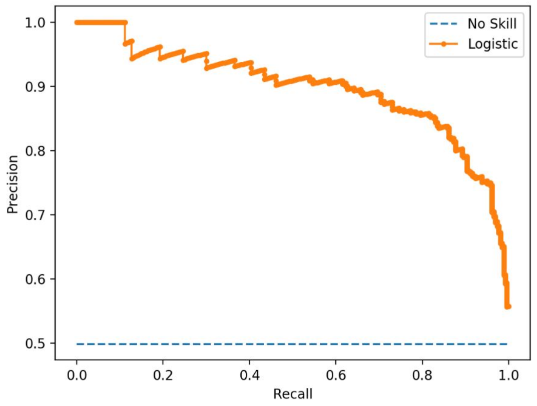
We could construct the ROC (receiver operating characteristic) curve in the following way. First, we depict a 2D plane in which the x-axis correspond to the true positive rate (TPR) and the x-axis to the false negative rate (FNR), which is defined as $\text{FNR} = 1 - \text{TNR} = \frac{\text{FN}}{\text{TN+FP}}$. Then, we scatter the points $(\text{FNR}(\tau)\ ,\ \text{TPR}(\tau))$ corresponding with the performance of the classifier for several values of the parameter $\tau \in (0, 1)$. The ROC curve is a curve that interpolates such points along with $(0, 0)$ and $(1, 1)$, an example of ROC curve can be seen in Figure (4a). See that the ROC curve of a random classifier would be a straight line joining $(0, 0)$ and $(1, 1)$, on the other hand, for an optimum classifier, it would consist of two straight lines connecting $(0, 0)$ with $(0, 1)$ and $(0, 1)$ with $(1, 1)$. Under these conditions, the optimum threshold would be the one that produces the closest point to $(0, 1)$.

The AUC ROC refers to the area between the ROC curve and the x-axis, the larger the AUC ROC, the better the performance of the classifier. See that the AUC ROC is a suitable metric for balanced data, i.e., for data in which the percentage of samples in the positive class is approximately 50%; however, it could be misleading for imbalanced data. Let us assume that the number of samples in the negative class is much larger than the samples in the positive one and the classifier is biased towards the negative class. The FN number will tend to be small since there are few positives and hence the probability of a wrong classification for negative predictions is reduced. For the same reason, the number of TN will be larger than the FP, therefore, the FNR will be close to 0 and the TPR close to 1. This system would lead to a ROC curve similar to the optimal one, and hence the AUC ROC would be close to 1, indicating an excellent classifier. To solve this issue, we should take into account a different kind of curve.

The PR (precision recall) curve is constructed in a similar fashion as the ROC curve, the only difference is that the y and x axis of the PR curve correspond to the precision and recall respectively. This curve has some interesting characteristics. If P and N are the number of positive samples and samples respectively, the PR curve of a random classifier is approximately a horizontal line with $\text{precision} = {}^{\text{P}}\!/_{\text{N}}$. This can be understood in the following way, for each sample, the random classifier returns a probability p of being in the positive class, with $p \sim U[0, 1]$.



(a) ROC curve



(b) PR curve

Figure 4: Empirical curves

Let us impose a threshold $\tau \in (0,1)$, such that the sample is considered positive if $p > \tau$, since $p \sim U[0,1]$, the number of positive predictions shall be approximately $N(1-\tau)$. These samples are a random subset of the original dataset, we must expect a true positive rate of approximately $\frac{\mathrm{P}}{\mathrm{N}}N(1-\tau)$, hence precision $= \frac{\mathrm{TP}}{\mathrm{TP+FP}} \approx \frac{N(1-\tau)\mathrm{P}/\mathrm{N}}{N(1-\tau)(\mathrm{P}/\mathrm{N}+1-\mathrm{P}/\mathrm{N})} = \mathrm{P}/\mathrm{N}$, for all values of τ. On the other hand, recall $= \frac{\mathrm{TP}}{\mathrm{TP+FN}} \approx \frac{N(1-\tau)\mathrm{P}/\mathrm{N}}{N((1-\tau)\mathrm{P}/\mathrm{N}+\tau\mathrm{P}/\mathrm{N})} = 1-\tau$. A non-random classifier would join the point[3] $(0,1)$ with $(1,0)$. See that, the closest the points to $(1,1)$, the better the classification. An example of PR curve is shown in Figure (4b). The AUC PR is simply the area between the PR curve and the x-axis. The PR is more suitable for imbalanced data than the ROC, since the precision only depends on TP and FP, hence, it is unaffected by classifiers biased toward the negative class.

We have seen some empirical measurements of the performance of a binary classifier. For multi-class and multi-label classifiers we may generalise the above metrics in different ways. We may consider that the performance of the classifier for each label is equivalent to the binary case, for a sample with label i, the prediction is positive if i is included in the predicted labels and negative otherwise. See that in the multi-class case there is only one predicted label and is the one with the largest probability. There are two main procedures for generalising the metrics to the multi-label and multi-class case:

- Macro-averaging: Consist in averaging the metric computed over each label. For instance, the precision ($P_i = \frac{TP_i}{TP_i+FP_i}$) of a two-class multi-label classifier in the macro-averaging scheme would consist in computing the mean of the precision for each of the labels, the resulting precision would read $P = \frac{P_1+P_2}{2} = \frac{TP_1(TP_2+FP_2)+TP_2(TP_1+FP_1)}{2(TP_1+FP_1)(TP_2+FP_2)}$.

[3]Choosing the point $(0,1)$ is a mere convention, see that this corresponds to $\tau \to 1$, where recall $\to 0$ and precision $\to \frac{0}{0}$

- Micro-averaging: Consist in considering the predicted labels as different outputs of a single binary classifier. Using the example of the precision of a two-class classifier, the precision would be $P = \frac{TP_1+TP_2}{TP_1+TP_2+FP_1+FP_2}$.

If the classes have approximately the same number of samples both averages shall be similar. In the case of an imbalanced dataset, the less frequent labels will have a larger contribution in the macro-average scheme than in the micro-average one. For this reason, if the result of the macro-average is worse than the one of the micro-average, it indicates that the classifier is performing worse in the labels with fewer occurrences. This characteristic is going to be recurrent in the computational models, hence, it is noteworthy to illustrate it with a simple example Let us assume that three classifiers (Π_i) predict three classes ($\mathcal{C}_i$) leading to the following results

Π_1	TP	FP
$\mathcal{C}_1$	900	100
$\mathcal{C}_2$	1	9
$\mathcal{C}_3$	1	9

Π_2	TP	FP
$\mathcal{C}_1$	100	900
$\mathcal{C}_2$	9	1
$\mathcal{C}_3$	9	1

Π_3	TP	FP
$\mathcal{C}_1$	500	500
$\mathcal{C}_2$	5	5
$\mathcal{C}_3$	5	5

We may compute the precision using micro-averaging and macro-averaging:

Classifier	Π_1	Π_2	Π_3
Micro-averaging	0.884	0.115	0.500
Macro-averaging	0.366	0.633	0.500

Table 4: Precision using different schemes

As we can see, if there is a wide difference between the performance of the model in different labels, the metrics can be misleading. We must take into account the importance of the less common labels before choosing a scheme for computing the metrics

3.5 Clustering

Clustering algorithms form part of a branch of machine learning known as unsupervised learning. In supervised learning we provide just sample inputs without sample outputs to the models. The algorithms learn from the data without any label looking for patterns or correlations between the inputs. The aim of clustering algorithms is to categorize the input into sets such that the elements that are included in the same group are similar to each other and different from the objects of other sets.

In general, the data is passed to the algorithm as an array of continuous features. The two main clustering algorithms are the k-means algorithm and the hierarchical clustering method.

3.5.1 k-means algorithm

The k-means algorithm is a clustering method based on forming sets associating a centroid to each cluster. The hyper-parameter k correspond to the number of clusters and it is fixed beforehand. The algorithm is initialized choosing k samples randomly as the centroids, although there are more sophisticated initialization methods that improve the convergence. After initialization, the samples are assigned to the cluster with the closest centroid. Iteratively, the algorithm recalculates the centroids based on the new assignments and reassigns the samples to the cluster with the closest centroid. The stopping criterion is based on the displacement of the centroids between consecutive iterations. See that, since the centroid corresponds to the geometric centre of the cluster, we may only obtain globular clusters. The pseudo-code for the k-means algorithms is shown below

Algorithm 1 k-means

```
1: Input ({ $\mathbf{x}_1, \cdots, \mathbf{x}_N$ }, $k$)
2: Initialize the centroids $\boldsymbol{\mu}_i$ with $k$ random samples
3: while Stopping criterion has not been met do
4:     for $m \leftarrow 1$ to $k$ do
5:         $\omega_m \leftarrow \{\}$
6:     for $n \leftarrow 1$ to $N$ do
7:         $j \leftarrow \arg\min_{j'} |\boldsymbol{\mu}_{j'} - \mathbf{x}_n|$
8:         $\omega_m \leftarrow \omega_j \cup \{\mathbf{x}_n\}$
9:     for $m \leftarrow 1$ to $k$ do
10:        $\boldsymbol{\mu}_m \leftarrow \frac{1}{|\omega_m|} \sum_{\mathbf{x} \in \omega_m} \mathbf{x}$
   return $\{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_k\}$
```

Let N be the number of samples, let k be the number of clusters, d is the number of features of each sample and i is the number of iterations, hence, the complexity of the algorithm is $\mathcal{O}(Ndki)$. The fact of being linear with respect to n makes it suitable for being applied to large datasets [47].

3.5.2 Hierarchical clustering

Hierarchical algorithms cluster the samples building a dendrogram that encodes the similarity between the inputs. There exist two hierarchical clustering methods:

- Agglomerative: Each input is initialized in a different cluster, at each step the algorithm merges a pair of clusters according to a linkage criterion.
- Divisive: All the inputs are assigned to the same cluster at the beginning of the algorithm, at each step a cluster is split in two based on a linkage criterion.

The linkage criterion is a measure of the distance between sets of points. It is based on a metric or a dissimilarity measure such as the euclidean distance or the cosine similarity, let us refer to this function as $d(\cdot,\cdot)$. Some common linkage criteria are the following:

Linkage criterion	Expression
Complete linkage	$\max\{d(a,b) : a \in A, b \in B\}$
Single linkage	$\min\{d(a,b) : a \in A, b \in B\}$
Average linkage	$\frac{1}{\lvert A\rvert \cdot \lvert B\rvert} \sum_{a \in A} \sum_{b \in B} d(a,b)$

Table 5: Linkage criteria

The pseudo-code for creating k clusters using the linkage criteria $l(\cdot,\cdot)$ and the agglomerative clustering is the following:

Algorithm 2 Agglomerative clustering

```
1: Input ({ $\mathbf{x}_1, \cdots, \mathbf{x}_N$ }, $k$, $l$)
2: for $j \leftarrow 1$ to $N$ do
3:     $c_j \leftarrow \{\mathbf{x}_j\}$
4: $C \leftarrow \{c_1, \cdots, c_N\}$
5: while $size(C) > k$ do
6:     $\{c_{\min,1}, c_{\min,2}\} \leftarrow \arg\min_{c_1,c_2 \in C} (l(c_1, c_2))$
7:     remove $c_{\min,1}$ and $c_{\min,2}$ from $C$
8:     add $\{c_{\min,1}, c_{\min,2}\}$ to $C$
   return $C$
```

The divisive algorithms have a large computational complexity $\mathcal{O}(2^N)$ [48], hence, they are not used in practice. In contrast, the complexity of the standard hierarchical agglomerative clustering, as shown in Algorithm (2), is $\mathcal{O}(N^3)$ [49]. Furthermore, there exist some improvements of the algorithms that reduce it to $\mathcal{O}(N^2)$, such as the SLINK [50] or the CLINK[51]. In any case, the k-means algorithm is more suitable for large datasets.

3.5.3 Determining The Optimal Number Of Clusters

There exist several metrics for measuring the quality of the clustering. In this section we will discuss two of the most frequently used metrics, the within-cluster distance and the silhouette score.

The within-cluster distance is mainly used for assessing k-means algorithms, it is defined as the half of the mean of the squared distance between points in a cluster normalized by the number of points in the cluster:

$$\mathcal{W}(\{\mathcal{C}_i\}) = \frac{1}{2}\sum_{l=1}^{k}\frac{1}{|\mathcal{C}_l|}\sum_{i,j\in\mathcal{C}_l}\left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|^2$$

Where $\{\mathcal{C}_i\}$ is the set of clusters, it can be shown that this metric is equivalent to the sum of the squared distance between each point of the dataset and the centroid of the cluster to which it belongs, which requires fewer computations:

$$\begin{aligned}\mathcal{W}(\{\mathcal{C}_i\}) &= \frac{1}{2}\sum_{l=1}^{k}\frac{1}{|\mathcal{C}_l|}\sum_{i,j\in\mathcal{C}_l}\left\|\left(\mathbf{x}^{(i)} - \mu_l\right) - \left(\mathbf{x}^{(j)} - \mu_l\right)\right\|^2 \\ &= \sum_{l=1}^{k}\frac{1}{2|\mathcal{C}_l|}\sum_{i,j\in\mathcal{C}_l}\left(\left\|\mathbf{x}^{(i)} - \mu_l\right\|^2 + \left\|\mathbf{x}^{(j)} - \mu_l\right\|^2 - 2\left(\mathbf{x}^{(i)} - \mu_l\right)\left(\mathbf{x}^{(j)} - \mu_l\right)\right) \\ &= \sum_{l=1}^{k}\sum_{i\in\mathcal{C}_l}\left(\left\|\mathbf{x}^{(i)} - \mu_l\right\|^2 - \left(\mathbf{x}^{(i)} - \mu_l\right)\frac{1}{|\mathcal{C}_l|}\sum_{j\in\mathcal{C}_l}\left(\mathbf{x}^{(j)} - \mu_l\right)\right) = \sum_{l=1}^{k}\sum_{i\in\mathcal{C}_l}\left\|\mathbf{x}^{(j)} - \mu_l\right\|^2\end{aligned}$$

We have used $\mu_l = \frac{1}{|\mathcal{C}_l|}\sum_{i\in\mathcal{C}_l}\mathbf{x}^{(i)}$. Usually, the k-means algorithm is computed using several initialisations for the same k and the resulting cluster is the one that minimises the within-cluster distance. This metric is also used to choose the optimum value of k heuristically for small ranges of k employing the elbow method. It is based on the fact that $\mathcal{W}(\{\mathcal{C}_i\})$ tends to decrease as we increase k, the graph of the variation of $\mathcal{W}(\{\mathcal{C}_i\})$ as a function of k tends to be similar to the one showed in Figure (5). The elbow method consists in choosing the value of k from which the within-cluster distance does not significantly decrease.

The silhouette method can be applied to both the k-means clustering and the hierarchical clustering. We must define some functions before introducing it. For any given point i belonging

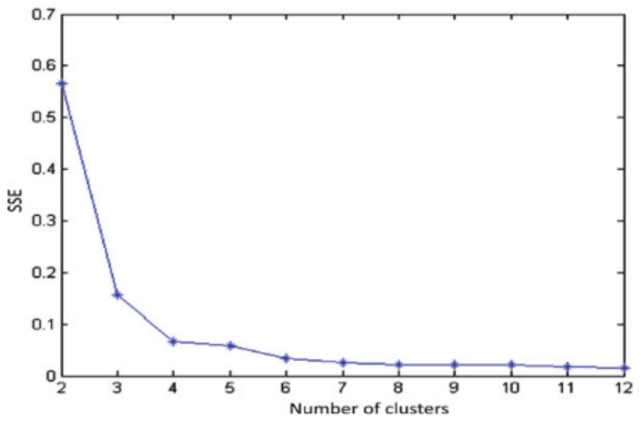


Figure 5: Example of variation of the within-cluster distance with respect to k. The optimum value could be $k = 4$. Figure taken from [52]

to $\mathcal{C}_l$, which includes more than one point, we define $a(i)$ as the mean distance between the points of cluster $\mathcal{C}_l$ and i:

$$a(i) = \frac{1}{|\mathcal{C}_l| - 1} \sum_{j \in \mathcal{C}_l} d(i, j)$$

On the other hand, we define $b(i)$ as the minimum distance between i and any point j such that $j \notin \mathcal{C}_l$:

$$b(i) = \min_{k \neq l} \frac{1}{|\mathcal{C}_k|} \sum_{j \in \mathcal{C}_k} d(i, j)$$

From the above functions, we may define the silhouette function of point i as:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |\mathcal{C}_l| > 1$$

and

$$s(i) = 0, \text{ if } |\mathcal{C}_l| = 1$$

See that $-1 < s(i) < 1$. The case $s(i) \to 1$ implies that $a(i) << b(i)$, which means that the point is correctly classified. On the contrary $s(i) \to -1$ implies that $b(i) << a(i)$, hence, the point is not properly classified. The silhouette score of the clustering is the mean of the silhouette functions of the points of the dataset.

3.6 Word embeddings

One fundamental problem in natural language processing (NLP) is how to teach a machine to read. For encoding the semantic significance we first need to transform the sentences into a format that is intelligible by the computer, in other words, we need to provide a numerical representation of the text. In NLP this numerical representation is usually applied to the words of a vocabulary.

The most simple representation of words is the one-hot encoding, let us assume that the vocabulary is a set of N words, we could assign a unique numerical label $i \in \{0, \cdots, N-1\}$ to each word, the one-hot encoding of a word with label i^* would consist in an N-dimensional array formed by 0 except for a 1 in the position i^*. This approach has several drawbacks, first of all, it is highly inefficient due to the sparsity of the array, in addition, all the words are equidistant, hence, the encoding does not contain any semantic information.

The disadvantages of the one-hot encoded vectors can be solved using word embeddings. The concept of word embedding arises from linguistics, it comes from the distributional hypothesis, which states that "words that occur in the same contexts tend to have similar meanings"[53]. Word embeddings are defined as dense arrays with dimension $m << N$ such that the representation encodes the similarity between words. The smaller the angle between embeddings, the greater the similarity between the words they represent, see that two words are similar if they tend to occur in the same context, for instance, the word 'dog' is more similar to 'cat' than to 'run'. It is also convenient that the embeddings show linear substructures between similar pairs of words, for instance, we would expect that $v(\text{France}) - v(\text{Paris}) \approx v(\text{Germany}) - v(\text{Berlin})$, where $v(\omega)$ is the embedding of the word ω.

The word embedding models are trained using a corpus, in other words, an extensive text, some common corpus for training word embeddings are formed by a large number of Wikipedia articles, news from Google News or tweets from Twitter. The vocabulary is the set of different words that appear in the corpus.

We will give an overview of the two techniques that we will use in our model to compute the word embeddings of the tags

3.6.1 Word2Vec

Word2Vec is an umbrella term that encompasses the embedding techniques introduced by T. Mikolov et al. in a series of papers in 2013 [54, 55].

The models, as presented in the first paper [54], are based on the previous work of Bengio et al. in 2003 [56], which consisted in predicting the next word given the previous C words in a sentence. It is illustrative to introduce the network sued by Bengio et al. It consisted on a

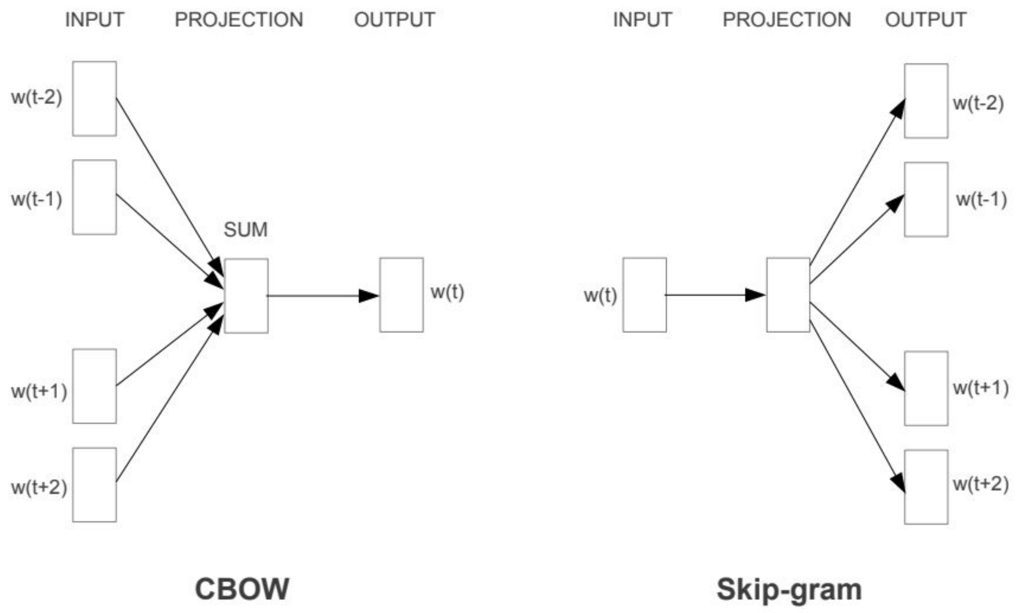


Figure 6: CBOW and skip-gram models. Figure taken from [54]

feedforward neural network formed by 4 layers. The first layer is the input layer, where the inputs are the C one-hot encoded vectors with dimension N, see that the input is not just one vector, as in Figure (2). The second layer is the projection layer, this layer is equivalent to a dense layer with multidimensional input, it transforms the output into a one-dimensional vector, the activation function is the identity. The third layer is a regular dense layer with dimension H and tanh activation function. Finally the output layer consist in N neurons with softmax activation function.

The first of the papers [54] introduced two different models for obtaining word embeddings. The continuous bag of words (CBOW) and the skip-gram. Both models are represented in Figure (6). Each model corresponds to a network with three layers. The inputs of each of these models are words represented as one-hot encodings $\mathbf{x} \in \mathbb{R}^N$, the hidden layer multiplies the input by the matrix $\mathbf{W} \in \mathbb{R}^{m \times N}$ and the result is passed the output layer. The output of each model has an interpretation; however we are not interested in it, the way in which the models produce the word embedding is via the matrix of the hidden layer. The matrix is $\mathbf{W} \in \mathbb{R}^{m \times N}$, the i-th column is the m-dimensional word embedding of the word whose one-hot encoding has a 1 in the position i, for short, we will refer to the latter word as the word with label i and we will denote it as ω_i. See that the bias terms are not included in any of the models

The aim of CBOW this model is to predict a word given the surrounding words in a sentence, we must choose the window size as a hyper-parameter, i.e., the number of words around the objective word that are used for the prediction, if the window size is C, the network would consider the previous and following $\lfloor \frac{C}{2} \rfloor$ words to the objective. For instance, in the sentence "Science is a way of thinking much more than it is a body of knowledge" using $C = 4$ on the objective word "thinking", the network would try to predict which word fits in this position based on the surrounding words $\{\text{way}, \text{of}, \text{much}, \text{more}\}$, on the other hand if we had chosen the first appearance of "is" the window would be reduced to three words $\{\text{science}, \text{a}, \text{way}\}$. The network of the continuous bag of words model is a modification of the one presented by Bengio et al. The input layer consists of the set of surrounding words of the objective, they are represented as one-hot encodings with dimension N. The hidden layer multiples each input $\mathbf{x}_i \in \mathbb{R}^N$ by a matrix $\mathbf{W} \in \mathbb{R}^{m \times N}$, averages the result and uses the identity as activation function, hence, the output of the hidden layer is $\mathbf{h} = \frac{1}{C}\mathbf{W}\left(\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_C\right)$. Finally, the output layer consists in N neurons with softmax activation function. See that component i of the output can be seen as $p(\omega_i|\omega_1, \cdots, \omega_C)$, in other words, the probability of predicting word ω_i if the word at the windows were $\{\omega_1, \cdots, \omega_C\}$. We want to maximize $p(\omega_O|\omega_1, \cdots, \omega_C)$, where ω_O is the actual objective word, since $p(\omega_O|\omega_1, \cdots, \omega_C) \in (0, 1)$, it is equivalent to minimizing the loss function $E = -\log(p(\omega_O|\omega_1, \cdots, \omega_C))$ [57].

The second model introduced in the paper is the one that is commonly used for obtaining word embeddings. It is the skip-gram model, we may consider it as the inverse of the CBOW. This model consists in predicting the surrounding words in a sentence based on the objective word. For instance, using again the sentence "Science is a way of thinking much more than it is a body of knowledge" with a window size of $C = 4$ on the objective word "thinking", the skip-gram model will try to predict the previous and following $\lfloor \frac{C}{2} \rfloor$ words. The input layer is just the one-hot encoding of the objective word $\mathbf{x} \in \mathbb{R}^N$. The hidden layer multiplies $\mathbf{x}$ by a matrix $\mathbf{W} \in \mathbb{R}^{m \times N}$ and uses an identity activation. The output layer produces C copies of the output of the hidden layer multiplied by the correspondent matrix $\mathbf{W'} \in \mathbb{R}^{N \times m}$ and applies a softmax activation [57]. The fact of using the same vector for representing all the prediction could seem outlandish; however, is the most suitable approach for training taking into account that we are just interested in the hidden layer. Alternatively, the skip-gram model can also be trained considering that the output encodes the window in a single vector, as if it were a multi-label classification problem with the particularity of using the softmax activation function. In this model we are interested in maximizing $p\left(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I\right)$. We may express the loss function of the skip-gram as $E = -\log p\left(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I\right) = -\sum_{i=1}^{C} \log p\left(w_{O,i} \mid w_I\right)$, where $\omega_{O,i}$ is the actual word in the position i of the window.

See that we may express $p\left(w_i \mid w_j\right)$ in terms of the parameters of the layers

$$p\left(w_i \mid w_j\right) = \frac{\exp\left(v_{w_i}^T v_{w_j}\right)}{\sum_{w=1}^{N} \exp\left(v_w^T v_{w_i}\right)} \tag{19}$$

Where v_{w_j} is the j-th column of the matrix of the hidden layer $\mathbf{W}$, $v_{w_i}^T$ is the i-th row of the matrix of the output layer **W'** and N is the size of the output vector, i.e., the size of the vocabulary. This implies that training this version of the skip-gram model is highly inefficient, since, given that all the word embeddings are included in the denominator, $\nabla p\left(w_i \mid w_j\right)$ is a vector with $N \cdot m$ components.

The second paper [55] introduces some improvements on the skip-gram model to facilitate the training. The paper proposes a modification of the loss function. We recall that the Word2Vec models are trained using a corpus, which can be seen as an extensive text. Let us define ω_j as the j-th word in the corpus, not in the vocabulary, where $j \in \{0, \cdots, T\}$ and T is the length of the corpus. We may express the loss function of the skip-gram model, evaluated on the corpus, as:

$$L(\theta)=-\frac{1}{T} \sum_{j=0}^{T} \sum_{\substack{-m \leqslant j \leqslant m \\ j \neq 0}} \log p\left(w_{t+j}(\theta) \mid w_t(\theta)\right) \tag{20}$$

The improvement consist in redefining the probabilities $p\left(w_i \mid w_j\right)$, which are originally given in Equation (19), as:

$$\log \left(p\left(w_i \mid w_j\right)\right)=\log \sigma\left(v_{w_j}^T v_{w_i}\right)+\sum_{w_l \in \mathcal{W}_{\mathrm{k}}} \log \sigma\left(-v_{w_l}^T v_{w_i}\right) \tag{21}$$

Where $\sigma(x)=\frac{1}{1+e^{-x}}$ and $\mathcal{W}_{\mathrm{k}}$ is a set of k words randomly sampled from a discrete distribution $P(\omega)$. The most suitable values of k are between 5 and 20 for small training sets and between 2 and 5 for large datasets. This redefinition is known as negative sampling, it reduces largely the amount of parameters used in $\log \left(p\left(w_i \mid w_j\right)\right)$. See that maximizing $p\left(w_i \mid w_j\right)$ implies maximizing $v_{w_j}^T v_{w_i}$ and minimising $v_{w_l}^T v_{w_i}$ $\forall w_l \neq w_i$ both in Equation (19) and Equation (21.

The optimal distribution $P(\omega)$ is found empirically. We must introduce the concept of unigram distribution of the corpus $U(\omega)$ as $U(\omega_i)=\frac{f_i}{N}$, where f_i is the number of times that word ω_i appears in the corpus. The optimal distribution $P(\omega)$ was found to be $P(\omega_i)=\frac{f_i^{3/4}}{N}$ [55], see that this distribution is halfway between the unigram distribution $U(\omega)$ and the uniform distribution $P(\omega)=\frac{1}{N}$.

Finally, the model requires a subsampling of frequent words, such as "in", "the" or "a", since they add less semantic information than the rare words. Before using the corpus, each word ω_i is discarded with a probability given by the empirical formula:

$$P(\omega_i) = 1 - \sqrt{\frac{t}{f_i}}$$

The parameter t is usually chosen as $t \approx 10^{-5}$.

It is easy to understand why the skip-gram model leads to similar embeddings for similar words. We recall that two words ω_i and ω_j are similar if they tend to occur in similar contexts, i.e., if switching between both words keeps the coherence of the sentence. Each time that words ω_i and ω_j appear in the corpus, they are passed as one-hot encodings, they are multiplied by $\mathbf{W}$ in the hidden layer and passed to the output layer. Hence, the input of the latter layer will be the corresponding column of the matrix of the hidden layer ($\mathbf{W}$). Moreover, if we assume that the sample outputs are alike, the network will train the parameters so that they become similar, hence, the word embeddings, which correspond to columns of $\mathbf{W}$, shall be similar.

The skip-gram model with negative sampling is a milestone in the field of NLP and has been used extensively in the last decade. Recently, it has been slowly replaced by more accurate algorithms, such as ELMo [58]. The reader may explore some examples of word embeddings in the TensorFlow embedding projector[4]

3.6.2 GloVe

GloVe is an algorithm for producing word embeddings using unsupervised learning based on co-occurrence of words [59]. It was created in 2014 at Stanford as an open-source project. This algorithm combines techniques from the two main approaches to word embeddings at the time of publication of the model. These approaches were the Matrix Factorization Methods and the Shallow Window-Based Methods. The Matrix Factorization Methods are based on the low-rank factorization of matrices that encoded statistical information of the co-occurrence of words in the corpus [60]. The Shallow Window-Based Methods are based on models that make predictions of words in a local window, the most notorious example of this approach is the Word2Vec model.

The basis of the model is the matrix of word-word co-occurrence X, we define it as an $N \times N$ matrix, where N is the size of the vocabulary, i.e., the set of unique words on the corpus. Let us set a window's size C, for each appearance of the word ω in the corpus, we define the context of ω as the $\lfloor \frac{C}{2} \rfloor$ previous and following words. The component X_{ij} counts how many times the word ω_i appears in the context of ω_j. See that X is a symmetric matrix. We denote as $X_i = \sum_k X_{ik}$ the

[4]`https://projector.tensorflow.org/`

Probability and Ratio	solid	gas	water	fashion
$P(k \mid \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k \mid \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k \mid \text{ice})/P(k \mid \text{steam})$	8.9	8.5×10^{-2}	1.36	0.96

Table 6: Co-occurrence probabilities for words 'ice' and 'steam'.
Example taken from [59]

number of times that any word appears in the context of word ω_i. We define $P_{ij} = P(\omega_j|\omega_i) = \frac{X_{ij}}{X_i}$ as the probability that word ω_j appears in the context of ω_i.

From the previously defined parameters, such as the word-word co-occurrence or the probabilities, we may obtain a scalar that represents the co-occurrence between two words. To obtain word embedding, we will include a third word ω_k. As we may see in the example depicted in Table (6), given a set of three words ω_i, ω_j and ω_k, if ω_i and ω_j are highly correlated with ω_k as in $\{\omega_i, \omega_j, \omega_k\} = \{$ice , steam, water$\}$ or not correlated at all as in $\{\omega_i, \omega_j, \omega_k\} = \{$ice , steam, fashion$\}$ then the ratio of probabilities verifies $P(\omega_k|\omega_i)/P(\omega_k|\omega_j) \approx 1$. This example inspired the authors of Glove to define the following function to obtain the embeddings:

$$F\left(w_i, w_j, \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \tag{22}$$

Where $w \in \mathbb{R}^m$ are the word embeddings, see that $\tilde{w}_k$ represent the same embedding as w_k. We may do some transformations on Equation (23) in order to impose the expected features of the embeddings. First of all, we expect that the embeddings preserve the linear substructures, for instance, we would expect that $w(\text{king}) - w(\text{man}) = w(\text{queen}) - w(\text{woman})$, so it would be convenient to impose that the dependency of (w_i, w_j) has the form $(w_i - w_j)$. Furthermore, since the right-hand side is a scalar and we are interested in the linear structures, it would be suitable that the argument of the function was a scalar, we may impose it taking the dot product, leading to a function of the form:

$$F\left(\left(w_i - w_j\right)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \tag{23}$$

We may impose a new restriction on $F(x)$ to solve Equation (23) easily. We may require that

$F(x)$ be a homomorphism between $(\mathbb{R},+)$ $(\mathbb{R}_{>0},+)$. Imposing this condition we obtain:

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F\left(w_i^T \tilde{w}_k\right)}{F\left(w_j^T \tilde{w}_k\right)} \tag{24}$$

On the one hand, $F(x) = e^x$ is a solution of Equation (24). On the other hand, combining Equation (23) and Equation (24) we may obtain a solution for $F(w_i^T \bar{w}_k)$:

$$F\left(w_i^T \tilde{w}_k\right) = P_{ik} = \frac{X_{ik}}{X_i} \tag{25}$$

From the above results, we may obtain an expression for the dot product $w_i^T \bar{w}_k$:

$$w_i^T \tilde{w}_k = \log\left(X_{ik}\right) - \log\left(X_i\right) \tag{26}$$

We have obtained an expression for $w_i^T \bar{w}_k$ imposing restrictions on the function $F(x)$. However, see that Equation (26) is inconsistent. The left-hand side should be invariant under exchange $w_i^T \leftrightarrow \tilde{w}_k$, $\log\left(X_{ik}\right)$ verifies this condition, since $X_{ik} = X_{ki}$; however, $X_i \neq X_k$. This inconvenience can be solved in the computational model including $\log\left(X_i\right)$ in a bias term. The authors of GloVe proposed the following model [59]:

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log\left(X_{ik}\right) \tag{27}$$

See that X has a significant amount of zero components, hence, the right-hand side term will diverge in most cases. Nevertheless, it is not a problem since we obtain the parameters $\{w_i^T, \tilde{w}_k, b_i, \tilde{b}_k\}$ from a loss function, which may be defined in a way that vanish all the divergent terms. In the GloVe model we define the loss function as a weighted MSE:

$$\mathcal{L}(\theta) = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \tag{28}$$

Where the function $f(x)$ verifies the following restrictions:

- In order to avoid divergences, $f(x) \to 0$ if $x \to 0$ faster than $\log^2(x)$ tending to $-\infty$.
- The larger the co-occurrence, the more robust the similarity between words. Hence, $f(x)$ must be non-decreasing to give more weight to pairs of words with high co-occurrence

- Some words appear much frequently than others, it would be suitable to avoid assigning large weights to these words, hence, we must impose $f'(x) \leq 0$

Following the above conditions for $f(x)$ we may define the weight function heuristically as [59]:

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{29}$$

Where $x_{\max} = 100$ and $\alpha = 3/4$. See that the input of the loss function in Equation (28) is just the matrix of word-word co-occurrence, we may obtain the word embeddings by gradient descent without making use of a neural network. In general, Word2Vec tends to outperform GloVe for corpus both in English [61, 62] and in foreign languages [63]. However, GloVe is easier to parallelize [64].

3.7 How to represent sound

Sound is a pressure wave generated by the vibration of the particles of the medium in which it propagates. The simplest way to store sound is through a waveform. In this context, a waveform is a one-dimensional array in which component i represents the amplitude of the sound wave at time $\frac{i}{n}$ from the beginning of the measurement. The variable n is the sample rate, which represents the number of measurements of the amplitude per second.

In practical applications, the sound is usually represented using spectrograms. To understand them, we may introduce the concept of Fourier transform. Let $f(t)$ be the waveform of the audio signal, spanned over the interval $[0, \Omega]$, the Fourier transform of $f(t)$ is defined as [65]:

$$\hat{F}\left[f(t)\right] = c_0 + \sum_{n=1}^{\infty} \left\{ c_n e^{i2\pi nt/\Omega} + c_{-n} e^{-i2\pi nt/\Omega} \right\}$$

If the signal is discrete $\{f(t_k)\}_{k=0}^{N}$, where $t_k = \frac{k\Omega}{N}$, then the coefficients c_n may be computed as:

$$c_n = \frac{1}{N} \sum_{k=0}^{N-1} f\left(t_k\right) e^{-i2\pi nk/N}$$

Since $f(t) \in \mathbb{R}$, $c_{-n} = \overline{c_n}$. The Fourier transform decomposes an arbitrary wave $f(t)$ into a superposition of simple waves, i. e., waves with constant frequencies n/Ω. The Fourier spectrum is defined as the array $\{2|c_n|^2\}$, the fundamental frequency c_0 is not taken into account since it corresponds to the background level, which is inaudible.

See that the Fourier spectrum does not capture the temporal variation of the signal, this drawback is solved using spectrograms. Spectrograms are based on computing the Fourier spectrum of the subsignals $\{f(t)\omega(t-\tau_m)\}_{m=1}^{M}$, where $\omega(t-\tau_m)$ equals 1 if $t \in [\tau_{m-1}+\delta, \tau_{m+1}-\delta]$ and 0 otherwise, the parameters δ and $\{\tau_m\}_{m=1}^{M}$ are arbitrary as long as they verify $\sum_{m=1}^{M} \omega(t-\tau_m) = 1$ $\forall t \in [0, \Omega]$. The spectrogram is a 2-dimensional array in which the x-axis represents time and the y-axis represents frequency. Given a value of the x-axis, t which verifies $t \in [\tau_{m-1}+\delta, \tau_{m+1}-\delta]$, the y axis depicts the values of the Fourier spectrum of $f(t)\omega(t-\tau_m)$ on their corresponding frequencies [65].

Spectrograms are represented using a linear frequency axis; however, humans perceive frequencies logarithmically, we can detect variations in the frequency of sounds more easily for low frequencies than for high frequencies. Empirically, it has been shown that we may perceive the pitch linearly up to approximately $1,000$ Hz and logarithmically from $1,000$ Hz [66]. Motivated by this fact, Steinberg [67] and Fletcher [68] presented a frequency scale suited to the human perception of Pitch. This empirical scale is known as the mel scale, there are different formulas to describe it, being the most popular one was presented by O'Shaughnessy [69]:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$$

Where f is the frequency in Hz. We may use this scale to define new kinds of spectrograms. A mel spectrogram is a spectrogram in which the frequency axis is represented using mel scale. We may convert the elements of the Fourier spectrum into decibels (dB) using the formula $dB(P) = 10\log_{10}\left(\frac{P}{P_0}\right)$, where P_0 is a reference value, usually $P_0 = 1$. We use the term log-mel spectrogram to refer to a mel spectrogram in which the values are measured in decibels.

4 Overview of the dataset

In this project we will employ the Million Song Dataset (MSD)[24], which provides metadata and audio features for a million songs, in addition, the 7digital library contains extracts of 30 seconds for each track in the MSD. We combine this dataset with the tags supplied by the last.fm dataset, it contains a correspondence between the id of the MSD and lists of tags, where 505,216 songs have at least one tag assigned. These tags were obtained by crowdsourcing, created voluntarily by users of the last.fm service [70]. It is worth mentioning that the tagging process consists of social tagging, which means that the user can provide any sentence as the label. This

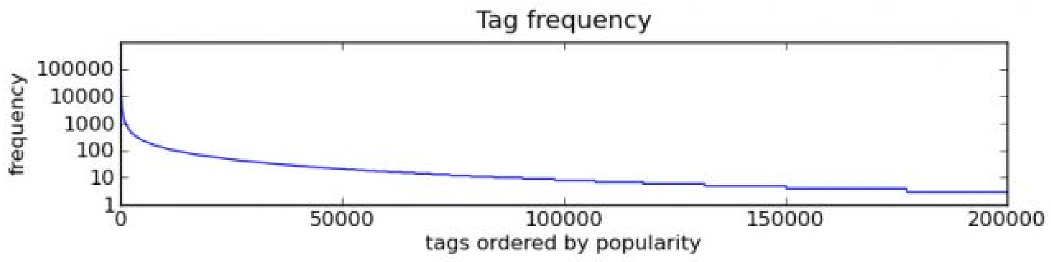


Figure 7: Distribution of tags in the last.fm dataset. Figure taken from last.fm

leads to several remarkable characteristics. First of all, there is a large number of tags, most of which have a significant intersection with other tags. Furthermore, since it comes from anonymous crowdsourcing, some tags are absurd or are inconsistent with the labelled track, moreover, some other tags are highly subjective. These attributes make the raw last.fm data set inadequate for tagging automatization, for this reason, researchers generally just use the 50 most frequent tags for classification, reducing the dataset to approximately 240,000 songs [26].

Let us get an overview of the last.fm dataset. In Figure (7) we show the distribution of tags, as we can see, they are highly imbalanced, we may consider that the frequency of a tag is an indicator of robustness, i.e., the most popular tags tend to be better suited to the track. The less popular tags tend to be nonsense or very specific tags. In Table (7) we show the 20 most popular tags.

Ranking	Tag	counts	Ranking	Tag	counts
1	rock	101,071	11	alternative rock	30,334
2	pop	69,159	12	jazz	30,152
3	alternative	55,777	13	beautiful	29,421
4	indie	48,175	14	singer-songwriter	27,910
5	electronic	46,270	15	metal	27,430
6	female vocalists	42,565	16	chillout	27,276
7	favorites	39,921	17	male vocalists	27,269
8	Love	34,901	18	Awesome	26,248
9	dance	33,618	19	classic rock	25,771
10	00s	31,432	20	soul	24,702

Table 7: Most popular tags in the last.fm dataset. See that the tags are case-sensitive

As we can see, there are some tags, such as “favorites” , “cool” or “awesome”, that are unsuitable for classification, since they do not correspond with any musical feature but with opinions. On another note, we may see that we may classify the tags in different groups that intersect between them, such as genres (rock, pop, etc.), voices (female vocalists, male vocalists) or dates (00s).

In Appendix (A.2) we have split the 100 most popular tags into groups of tags that correspond to the same category, the resulting categories are: genre, instruments, dates, voices, emotions, nationalities, meaningless and miscellaneous. It is noteworthy to mention that we have created the group "Meaningless" to encompass all the tags that are unsuitable for classification for being subjective, such as "Favourite", "cool" or "Awesome". A breakdown of the number of tags in each group can be seen in Table (8). These categories have been created classifying the 100 most popular tags of the last.fm dataset by hand.

Group	Genres	Instruments	Dates	Voices	Emotions	Nationalities	Meaningless	Miscellaneous
Total counts	961, 402	29, 539	136, 983	110, 457	260, 768	63, 068	260, 850	116, 245
Average counts	22, 358	14, 769	19, 569	22, 091	15, 339	15, 767	16, 303	16, 606

Table 8: Groups of tags based on the 100 most popular tags

5 Computational Model

In this project, we will reproduce the two models presented by J. Pons et al. [71]. These models follow an end-to-end architecture, in other words, for prediction purposes, the data is passed in raw form, without previous preprocessing, and the model generates the output in the target format. We will use the model on tags as a baseline.

The models can be decomposed in front-end and back-end, the latter is shared by both models but the former differs. The first model uses waveforms as input, it is characterised by the fact that no domain knowledge in music is used, the network learns all the musical features from scratch without any aid from the design of the model. The second model works with spectrograms, the front-end applies independently a series of vertical and horizontal filters on the raw spectrogram aimed to capture spectral and temporal features. This model uses domain knowledge for designing the filters. For instance, some filters span 7 units in the temporal scale, this is a convenient size for capturing kick-drums, whose time-frequency pattern is 7×38.

Figure (8) shows the scheme of the model. The waveforms are processed using a series of convolutional layers with dimension 3×1, it has been shown empirically that this dimension is suitable for networks that do not make use of domain knowledge [12]. The number of filters in

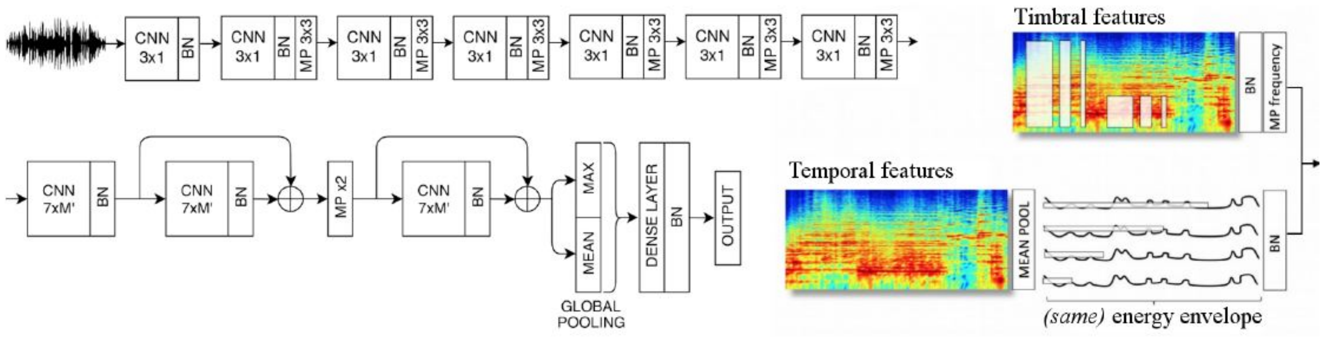


Figure 8: Model scheme, figure taken from [71]. Right: Spectrogram front end. Upper left: Waveform front end. Bottom-left: Backend. Acronyms: MP-Max pooling; BN - Batch Normalization; M'- number of channels of the input of the layer; CNN-Convolutional layer.

each convolutional layer is 64, 64, 64, 128, 128, 128 and 256 respectively.

Regarding the spectrograms, the audio tracks are converted into log-mel spectrograms using M mels, in the paper they use $M = 96$. The frontend applies two different filters to the spectrogram: the timbral filters and the spectrogram filters. The timbral filters are concatenations of convolutional layers, batch normalization and max pooling. These filters are applied independently, the number of filters and shapes are (F,$7 \times 0.9M$), ($2F$,$3 \times 0.9M$), ($4F$,$1 \times 0.9M$), (F,$7 \times 0.4M$), ($2F$,$3 \times 0.4M$) and ($4F$,$1 \times 0.4M$), see that in the frequency dimension we are actually computing the floor function. The number of filters (F) is an arbitrary parameter, in the paper they use $F = 16$, see that the first layer of the CNN captures simple patterns and the subsequent layers capture more complex features combining the foregoing patterns, hence, we will use $F = 32$ to increase the number of detected features. On the other hand, the temporal filters are not applied on the raw spectrogram but on the energy envelope, i.e, the result of applying a 1D average pooling with length M on the frequency axis. We apply several convolutional layers independently to the resulting array, whose shapes are 165, 128, 64 and 32. with the number of filters and length equal to (F,165), ($2F$,128), ($4F$,64) and ($8F$,32). The output of the frontend is the concatenation of the results of every filter.

The Backend contains three convolutional layers with 512 filters and two dense layers; the number of neurons of the first layer is arbitrary, in the paper they use $n = 500$, in the second layer, the number of neurons equals the number of classes in the classification problem. In Figure (8), the symbol $\bigoplus$ indicates addition, where padding is used to equalize the dimensions, the global pooling layer consists of an average pooling and a max pooling on the same input, the latter arrays are concatenated and flattened to use them as input in the layers, where the number of neurons equal to the number of classes (tags) in which we classify the tracks.

In addition, we use a 50% dropout before the dense layer. The activation function of all layers is ReLU, except for the output layer which is sigmoid. The models are trained using the Adam optimizer of SGD with a learning rate $\eta = 0.001$. Since the network is a multi-label classifier, we use binary cross-entropy as the loss function. We shall split the dataset in 80% training, 10% validation and 10% test.

We shall use the AUC ROC and AUC PR on the micro-averaging and macro-averaging schemes for assessing the performance of the models. In the paper, it is not specified if the authors use the micro-averaging or the macro-averaging scheme for computing the metrics.

We will use the Tensorflow Python API in all the experiments. Since the code is extensive, it is not included in the report but it is stored on a GitHub repository[5]. We may divide the code into two parts: preprocessing of the data and computational models. The preprocessing and the structure of the code for training are mainly based on another repository[6]. We have also uploaded to the repository the arrays containing the metrics of each model.

6 Models on tags

6.1 Model 1: Classification on raw tags

6.1.1 Setup

We may consider this model as a starting point, we will train the wave-front and spectrogram classifiers on the tags without filtering, for this purpose we will just consider the 100 most popular tags, discarding the rest. We chose such a large number, compared with the common choice of 50 tags, because we will reduce gradually the number of tags using different filtering mechanisms along our models. The set of tags that corresponds to the classes of the classifier is presented in Appendix A.1. This subset of 100 tags encompasses $1,929,985$ counts, which corresponds to 22.4452% of the counts of the original dataset. This model does not aim to be a comparison with the results of J. Pons et al., since the number of tags is different and their method for averaging the metrics is unknown. Nevertheless, the model architecture is identical to the one used by J. Pons et al., except for the number of filters in the spectrogram frontend.

We shall measure the performance of the models using both the macro-averaging and the

[5]`https://github.com/carlorop/Music_Tagger_on_Word_Embeddings`
[6]`https://github.com/pukkapies/urop2019`

micro-averaging scheme for assessing the models. We recall that in the macro-averaging scheme mode the metrics are computed for each label and the result corresponds to the mean of these metrics, in the micro-averaging the metrics are computed from a unique confusion matrix which is the sum of the confusion matrices for each label. Since the dataset is largely imbalanced, the less frequent tags will almost not affect the performance in the micro-averaging scheme, some tags are subjective and therefore they will lead to a poor within-label metric, reducing the macro-averaging measure independently of the goodness of the model.

6.1.2 Results

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.3090	0.8449	0.2278	0.7974
Spectrogram	**0.3253**	**0.851**	0.2387	0.8048

Table 9: Results of baseline model

The results are shown in Table (27), we have trained the model using 30 epochs. See that the dataset is imbalanced and the tags with a lower number of counts tend to be misclassified, this feature causes the divergence between the Micro-averaging and Macro-averaging metrics. It is noteworthy to mention that we will compute the AUC PR and AUC ROC for each model; however, our comparisons will be based on the AUC since, as we discussed in Section 3.4, the AUC ROC may be misleading for imbalanced datasets. See that the less frequent tags have more weight in the macro-averaged metrics than in the micro-averaged ones. Since the macro-averaged metrics are consistently worse than the micro-averaged ones, we may confirm that the less frequent tags tend to be more unsuitable for classification than the more frequent tags, either because they are subjective or because they do not match the track.

On another note, the models based on spectrograms show better performances than the ones based on waveforms. It is reasonable, since the former uses domain knowledge, using filters that capture certain musical patterns, on the other hand, the waveform model uses filters whose shape does not correspond to any particular musical structure. See that, hereinafter, we use the term spectrogram to refer to log-mel spectrograms.

In Table (10) we show the breakdown of the average AUC PR per tag group for the macro-averaged model with spectrograms. The group with the larger performance is "Genres", which is reasonable due to the design of the filters of the spectrogram model. On another hand, the group with the worst performance is “Meaningless”, as we expected due to the unsuitability of this kind of labels for classification problems.

Group	Genres	Instruments	Dates	Voices	Emotions	Nationalities	Meaningless	Miscellaneous
Average AUC PR	0.367	0.216	0.209	0.213	0.107	0.075	0.069	0.214

Table 10: Average AUC PR per group of tags for the baseline model

6.2 Model 2: Classification on filtered tags

6.2.1 Setup

As we have seen in the previous model, the tags with the lowest performance either are based on features that are unrecognizable for the classifier or are completely subjective. Therefore, in this model we train the classifier on a set of filtered labels. The set of tags remaining after filtering is showed in Appendix (A.3). This first filtering consists in removing the tags that are included in the groups “Meaningless” and “Nationalities” since the performance on these groups is significantly worse than the rest of the categories. Such tags are shown in Table (11).

favorites	Awesome	sexy	catchy	Soundtrack
USA	UK	british	american	heard on Pandora
seen live	cool	Favorite	Favourites	favourite
favorite songs	amazing	good	epic	Favourite Songs

Table 11: Set of tags removed from the dataset. See that the dataset is case sensitive

These labels correspond to highly subjective tags which cannot be recognised from musical features. See that we have not removed the tags associated with emotions since, even when these tags are subjective, they could be related to musical features. For instance, the tag “happy” could arise in songs with a fast tempo.

From the original 100 tags, we have removed 20 tags. The resulting tags account for $1,606,067$ counts, which represent 18.678% of the total counts of the dataset.

6.2.2 Results

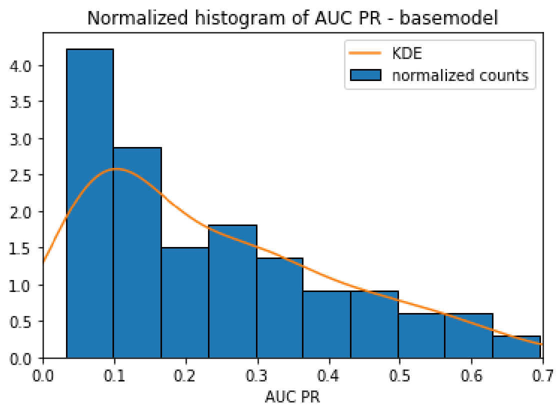
Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.3446	0.8642	0.2726	0.8288
Spectrogram	**0.3659**	**0.8718**	0.2836	0.8359

Table 12: Results of baseline model on filtered tags

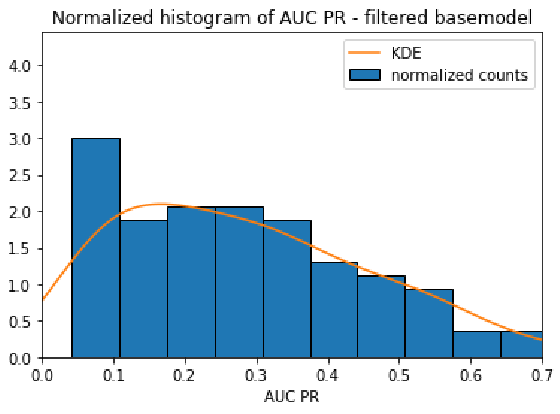
In Table (12) we may see the results of this model, trained on 30 epoch. As we expected, it outperforms the baseline model without filtering. Furthermore, it preserves all the trends that we saw in the previous model, the performance of the spectrogram model is larger than in the model with waveform and the micro-averaging scheme outperforms the macro-averaging scheme.

In Figure (9) we may see a comparison of the AUC PR per label between the baseline models with and without filtering. To provide an accurate comparison, since the number of tags is different, we have normalized the histograms. In addition, we have plotted the kernel density estimation (KDE), which is the empirical estimation of the density of the AUC PR. As we may see, the words from the groups “Meaningless” and “Nationalities” induce a peak in the KDE for small values of the AUC PR, which is another indicator of the fact that the performance in each tag of these groups is, on average, worse than in the tags of the rest of the groups.

Finally, we may see that the AUC PR per group in Table (13), the results are similar to the unfiltered model showed in Table (10).



(a) Baseline model



(b) Baseline model with filtering

Figure 9: Normalized histograms of the AUC PR per tag for the baseline models

Group	Genres	Instruments	Dates	Voices	Emotions	Miscellaneous
Average AUC PR	0.375	0.220	0.211	0.224	0.107	0.214

Table 13: Average AUC PR per group of tags with filtering

7 Models on clusters based on word embeddings

7.1 Introduction

Crowdsourcing is a common technique for obtaining data; however, it has major drawbacks in the case of obtaining datasets for classification, especially in the users may introduce their own labels. One of them is that some tags are highly subjective, another common issue is the large number of labels, which is impractical for classification, furthermore, there may exist several different labels that refer to the same concept. For this reason, in the last.fm dataset the classification is usually performed on a small subset of the labels, therefore, most of the tags of the dataset are discarded, losing a large amount of relevant information about the samples. In this model, we will propose a method to retrieve information from the less popular tags. This model consists in classifying the samples into classes that contain several similar tags. It is based on clusterings of word embeddings. In this way, we may create labels that encompass tags with similar meanings, such as "Allegro" and "Fast paced" or genres and their subgenres.

First, we assign a word embedding to each tag in the dataset. See that the smaller the angle between embeddings is, the more similar the tags are. On the other hand, the semantic information does not depend on the length of the embeddings, for convenience, we will normalize the word embeddings before clustering. Let us define the similarity measure $d(\mathbf{x},\mathbf{y})$ as $d(\mathbf{x},\mathbf{y}) = 1-\cos(\widehat{\mathbf{x},\mathbf{y}})$, where $(\widehat{\mathbf{x},\mathbf{y}})$ is the angle between the word embeddings $\mathbf{x}$ and $\mathbf{y}$. See that the smaller $d(\mathbf{x},\mathbf{y})$ is, the more similar the words are. Furthermore, it verifies $d(\mathbf{x},\mathbf{x}) = 0$ and $d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x})$. The similarity measure does not verify the triangle inequality; however, it is not necessary for clustering. We may show that $d(\mathbf{x},\mathbf{y})$ is equivalent to the squared euclidean distance, which is a desirable property for clustering. Assuming that vectors $\mathbf{x}$ and $\mathbf{y}$ are normalized:

$$\|\mathbf{x}-\mathbf{y}\|^2 = (\mathbf{x}-\mathbf{y})^\top(\mathbf{x}-\mathbf{y}) = \mathbf{x}^\top\mathbf{x} - 2\mathbf{x}^\top\mathbf{y} + \mathbf{y}^\top\mathbf{y} = 2 - 2\mathbf{x}^\top\mathbf{y} = 2d(\mathbf{x},\mathbf{y})$$

Since the dataset is large, we will use a slight variation of the k-means clustering algorithm normalizing the centroids. We recall that this algorithm was based on assigning each point to the cluster with the closest centroid. Hence, using the euclidean distance with normalized centroids is equivalent to the similarity measure $d(\mathbf{x}, \mathbf{y})$, since they induce the same order relations. For instance, given a point $\mathbf{x}$ assigned to the cluster with normalized centroid μ_{i^*}, it verifies $\|\mathbf{x} - \mu_{i^*}\| \leq \|\mathbf{x} - \mu_i\|$ for all centroids μ_i, since the norms are positive, the latter relation implies $\|\mathbf{x} - \mu_{i^*}\|^2 \leq \|\mathbf{x} - \mu_i\|^2$ hence $d(\mathbf{x}, \mu_{i^*}) \leq d(\mathbf{x}, \mu_i)$.

Using the variation of the k-means, we will group the tags into k clusters, which will be the classes of our classification problem. We would expect that, from the semantic information carried by the tags, each cluster captures at least a feature that can be detected in the tracks. In this way, we may increase the amount of information retrieved from the dataset, since we will not classify into tags but into their meanings, and therefore we will also retrieve information from tags with fewer counts that otherwise we would have rejected.

7.2 Model 3.1: Model on raw clusters

7.2.1 Setup

In this model we will create the word embeddings using a pre-trained model. This model was computed using the skip-gram Word2Vec algorithm over the English Wikipedia corpus[7]. The resulting vectors have 250 components and are normalized. If a word is not found in the corpus it is mapped to a zero vector and the composition of two words is computed as the normalized sum. Furthermore, it is case sensitive, like the tags of the last.fm dataset.

Regarding the clusters, we will group the tags in 500 clusters. We will include more tags than in the previous models; however, we will use only the tags with more than 100 counts. We consider that when the counts are fewer than 100 the tags tend to be incorrect or nonsense. [8]. This filtering removes 38.5836% of the total counts of the dataset but 98.5318% of tags.

It is convenient to inspect the data to assess the quality of the embeddings, to do so, we will reduce the dimensionality of the 250-dimensional using PCA. Given a set of vectors with dimension D, the PCA algorithm computes its linear projection in a hyper-plane with dimension d selecting the hyper-plane that maximizes the variance of the projected data. In Figure (10) we may see the projection of the word embeddings of the dataset of tags into a three-dimensional space. This projection captures 41.5% of the variance of the embeddings. We have used the tags with more than 100 counts. The projection shows a conic shape, this indicates that the dataset is skewed, in other words, a large number of words are close to each other. In addition, there is a

[7]`https://tfhub.dev/google/Wiki-words-250-with-normalization/2`

[8]The reader may verify this assumption looking at the complete list of tags and counts in plain text

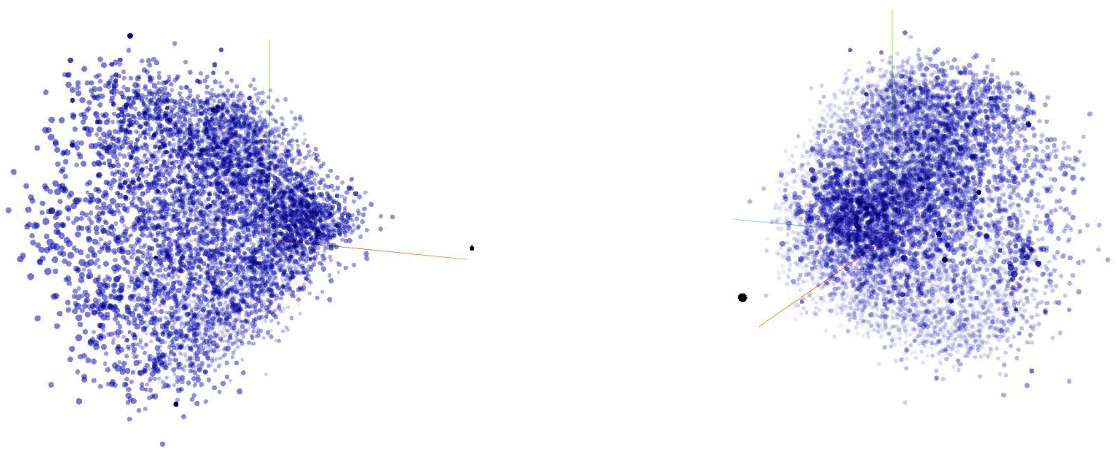


Figure 10: Three-dimensional projection of the set of word embeddings

word significantly separated from the rest of the samples, this word is “jjclscslst”. Surprisingly, this word appears in the English Wikipedia; however, it has no meaning at all. The fact that “jjclscslst” appears more than 100 times in the dataset gives an idea of its magnitude.

We may use the projection to inspect the neighbours of a word. In Table (14) we show the ten closest tags to “jazz”. The five closest words are the composition of “jazz” and a word that does not exist in the Wikipedia corpus, hence it is mapped into a zero vector, therefore the embeddings for these tags are the same as the one of “jazz”. These words could correspond to typos, as in “clasic” or “kappe”[9]; to slang, as in “fonky”; to other languages, as in “lekkere” which means “delicious” in Dutch; or to nonsense words, as in “cwcafe”. We may see that, when “jazz” is composited with an existing words, the closest embedding correspond to its composition with the words “blues” or “funk”. This fact indicates a suitable embedding since “blues” and “funk” are closely related with “jazz”. Jazz has its roots in blues and funk is a combination of jazz, soul and R&B.

A more rigorous empirical method to study the suitability of the word embedding algorithm would be to look for linear substructures. In Figure (11) we show the two-dimensional projection of the embeddings of 6 different pairs of nations and capital cities. We may see a linear substructure that captures the relation between nations and their capitals, in other words, if N_i, C_i and $v(\cdot)$ represent a nation, its capital and a word embedding respectively, then $v(N_1) - v(C_1) =$

kappe jazz	fonky jazz	clasic jazz	lekkere jazz	jazz cwcafe
jazz blues	jazz-blues	blues jazz	jazz funk	jazz-funk

Table 14: Closest tags to “jazz”

[9]This tag probably refers to the jazz musician Christian Kappe

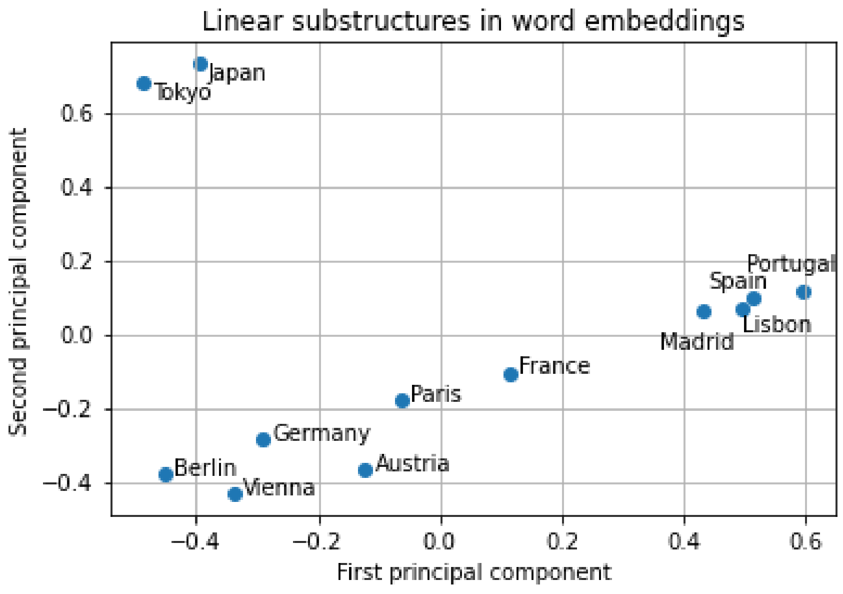


Figure 11: Example of linear substructures in word embeddings

$v(N_1) - v(C_1)$. Since the PCA applies a linear projection, the linear substructures are preserved. Furthermore, there are other features that indicate that the embeddings are suitable. For instance, the European nations tend to be further from the word "Japan" than from the rest of European nations. Furthermore, the pair of Germanic nations {"Germany","Austria"} are very close, the same happens with the Iberian nations {"Spain","Portugal"}. Moreover, "France" is midway between "Germany" and "Spain".

We have seen that the word embeddings capture the semantical meaning of the tags. Therefore, we will use the clustered embeddings as the classes of our classification problem. The clustered tags account for 5, 280, 968 counts, which represent 61.4163% of the total counts of the dataset, see that this is a large improvement with respect to the classifier on the most popular 100 tags, which only encompass 22.4452% of the counts.

7.2.2 Results

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.2097	0.8523	0.08102	0.7153
Spectrogram	**0.216**	**0.8546**	0.08439	0.7228

Table 15: Results of the model on raw clusters

Cluster	rock	chillout	alternative	pop	indie rock
Counts	140,408	120,534	96,805	88,086	81,353
AUC PR	0.5603	0.4863	0.3930	0.4435	0.3697
Position according to AUC PR	4	11	21	14	24
AUC PR in the baseline model	0.5575	0.2425	0.3389	0.4483	0.2992

Table 16: AUC PR on the 5 most popular clusters. The clusters are represented by its most popular tag. The comparison with the baseline model on tags is made on the most popular tag of each cluster

From now on, all the models will be trained using 20 epochs, due to the large training time that each model takes.

The model can be seen in Table (15). It shows some interesting features. There is a large difference between the micro-averaging and macro-averaging scheme, much larger than in the models with tags, which indicates two characteristics. First, there is a large imbalance between the counts of the clusters, in addition, the clusters with the largest number of counts tend to perform better than the ones with fewer counts. To confirm this hypothesis, we have computed the number of counts in each cluster. In Table (16) and Table (17) we show the five most and less popular clusters respectively, as well as their AUC PR and the ordering according to their performance. As we can see, the most popular clusters tend to reach better performances than the less popular ones. In addition, we may see that the less popular clusters correspond to useless tags. For this reason, in the following models we will filter the impractical clusters.

In Table (16) we may see a comparison of the AUC PR of the model on clusters and the baseline model for the most popular tag of some clusters. We may see that in some cases the performance is very similar, in particular in “rock” and “pop”. In the case of “alternative”, the cluster that it represents is formed by the addition of the word “alternative” to different genres, such as “alternative rock”, “alternative pop” or “alternative metal”, therefore, it encompasses different structures that would fit with the label “alternative” and that could be considered as

Cluster	JT	cha cha cha	Red Hot Chili Peppers	dum dum dum	folksy
Counts	121	129	145	177	184
AUC PR	0.0014	0.0003	0.0066	0.0065	0.0008
Position according to AUC PR	493	500	463	464	497

Table 17: AUC PR on the 5 less popular clusters. The clusters are represented by its most popular tag. The position is measured with respect to 500 clusters

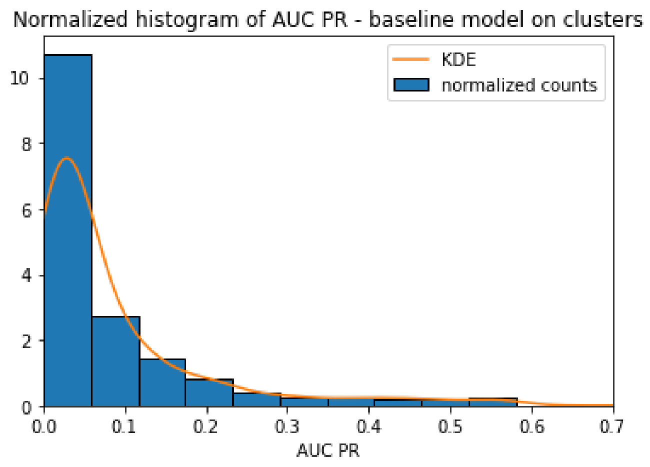


Figure 12: Normalized histogram of the AUC PR of the model on raw clusters

misclassifications in the model on tags, for instance, if a track has label "alternative rock" but not "alternative" then the prediction of "alternative" is a false positive. The same reasoning applies to the cluster "indie rock". In the cluster "chillout" we may find labels such as "electronica", "techno" or "downtempo", even though these labels could seem completely unrelated to the relaxing sounds of chillout music, this cluster seems to have encompassed all kinds of music made with electronic instruments. This hypothesis is coherent with the large improvement of AUC PR with respect to the baseline model on tags, since it would solve the problem of misclassifying different genres of electronic music that contain similar sounds.

In Figure (12) we may see the normalized histogram for the AUC PR. The KDE is highly skewed towards the small values of the AUC PR. As we expected, most of the clusters are unsuitable for classification, therefore, in the following models we will reduce the number of clusters.

7.3 Model 3.2: Grid-search on hyper-parameters

7.3.1 Setup

In the previous model, the large number of tags included in the cluster led to a deficient classification. We will improve the previous model removing clusters according to the filtering of Model 2. We shall remove all the clusters that do not contain any tag from the filtered 100 most popular tags, i.e., the 80 tags showed in Appendix A.3. Furthermore, we will select the best set of hyper-parameters using the grid-search technique. We recall that a hyper-parameter is a parameter of the model that cannot be trained in the learning process. The grid search technique consists in comparing the performance of the model for all possible combinations of the hyper-parameters, selecting the combination that maximizes the performance. See that if the model has n hyper-parameters $\{\lambda_i\}_{i=1}^n$ such that $\lambda_i \in \mathcal{C}_i$, then the number of combinations is $\prod_{i=1}^n |\mathcal{C}_i|$, hence, this technique tends to be computationally expensive.

The embedding algorithm will be one of the hyper-parameters of the model. We will use three different pre-trained embeddings: The skip-gram Word2Vec from Model 3.1, the GloVe algorithm trained on the English Wikipedia corpus with 200 components[10] and the GloVe model trained on the Twitter corpus using 200 components[11]. Moreover, we will tune the threshold for removing tags from the dataset before the clustering. In Model 3.1 we removed all the tags with fewer than $n = 100$ counts, for the grid search we shall consider that this hyper-parameter may take two values $n \in \{100, 500\}$. See that this first step of the clustering is applied on the raw tags, the labels that appear less than n times in the tracks are not included in the clusters.

Since the purpose of the grid-search is optimizing the choice of hyper-parameters we will just compute it on the spectrogram model since this model leads to the best performances in both schemes.

7.3.2 Results

Model	Micro-averaging				Macro-averaging			
Threshold / Algorithm	100		500		100		500	
	AUC PR	AUC ROC	AUC PR	AUC ROC	AUC PR	AUC ROC	AUC PR	AUC ROC
Word2vec	0.3928	0.8466	0.3881	0.8567	0.3216	0.8203	0.3228	0.8241
Glove Wikipedia	**0.4010**	0.8571	0.3962	0.8632	0.3228	0.8241	0.3107	0.8112
Glove Twitter	0.3958	**0.8580**	0.3887	0.8565	0.3105	0.8068	0.3116	0.8191

Table 18: Results of the grid search using $k = 500$ clusters and log mel spectrograms as input

The results of the model are showed in Table (18). For each combination of hyper-parameters, the model on clusters outperforms the model on 80 filtered tags, whose results can be seen in Table (12). We will discuss this fact in depth in the following model, which will consist in tuning the number of clusters.

[10] `https://nlp.stanford.edu/data/glove.6B.zip`

[11] `https://nlp.stanford.edu/data/glove.twitter.27B.zip`

The performance of the models is very similar for every combination of the hyper-parameters, the difference in the performance is not significant enough to consider that one particular combination leads to better performances than the rest. However, we will consider that the best combination is the one that maximizes the AUC PR, in this case, the model with embeddings generated by the Glove algorithm trained on the Wikipedia corpus with $n = 100$.

7.4 Model 3.3: Tuning the number of clusters

7.4.1 Setup

Originally, we included the number of clusters in the grid-search using two different values of k; however, during the training of the models we realized that the number of clusters was the hyper-parameter with the largest influence in the results, hence, it is convenient to widen the range of k for the hyper-parameter optimization. In this model, we will tune the number of clusters in the range $k \in \{100, 250, 500, 750, 1000\}$, we shall tune just this hyper-parameter on the Glove model with $n = 100$. See that if we had included it in the grid-search with the extended range, it would have led to 30 different combinations. Taking into account the computation time and the number of models trained in this project, it would have been unfeasible to train such a high number of models for this section.

It is noteworthy to mention that the number of clusters k refers to the number of clusters before filtering them removing all the clusters that do not contain any of the 80 filtered tags showed in Appendix A.3.

Once we tune k, we will use the optimum set of hyper-parameters to compare the model with clusters of word embeddings with the baseline model. For a fair comparison, we will run the baseline model just on tags that are the most popular tag of its cluster. See that this is not a like-for-like comparison, since we are comparing different models. Clustering the tags creates labels that are characterized by more general features than in the case of raw tags.

7.4.2 Results

Number of clusters (k)	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
100	**0.4225**	0.8009	0.3661	0.7640
250	0.4013	0.8370	0.3909	0.7995
500	0.3928	0.8466	0.3122	0.8081
750	0.3909	0.8550	0.3096	0.8173
1000	0.3895	**0.8606**	0.3138	0.8242

Table 19: Tuning of the number of clusters before filtering

The results of the hyper-parameter tuning show a contradicting trend in the AUC PR and AUC ROC, to explain this feature, we must inspect the confusion matrix. We recall that such a matrix represents the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) predictions of the classifier. The aforementioned metrics depend on the choice of threshold (τ). The threshold is the minimum probability predicted in a class which leads to a positive prediction. We will compare the confusion matrix of the classifiers for $k = 100$ and $k = 1000$, since these classifiers maximize the AUC PR and AUC ROC respectively. Since the number of samples is different in both models, we will compare the confusion matrix divided by the number of samples used in each classifier.

In Table (20) and Table (21) we show the confusion matrix for the clusters representing "rock" and "chill" respectively for both classifiers using three different thresholds. We have selected these clusters because we expect that "rock" is defined by recognisable features whilst "chill" is more subjective. We may see some common trends. First of all, the number of true negative samples is significantly larger than any other category, this is because the negatives of a label refer to any other label in the classification, clearly, the number of counts for any label is much less than the

k / Threshold	100				1000			
	TP	TN	FP	FN	TP	TN	FP	FN
0.33	0.2498	0.4821	0.1954	0.0725	0.1853	0.5559	0.1627	0.0954
0.50	0.1819	0.5765	0.1011	0.1403	0.1060	0.6550	0.0636	0.1752
0.67	0.0746	0.6507	0.0269	0.2477	0.0309	0.7076	0.0110	0.2503

Table 20: Normalized confusion matrix for the label "rock" for the models on clusters

Threshold \ k	100				1000			
	TP	TN	FP	FN	TP	TN	FP	FN
0.33	0.0134	0.8579	0.032074	0.096507	0.002232	0.933200	0.005186	0.0593
0.50	0.0019	0.8870	0.0030	0.1080	0.0002	0.9379	0.0004	0.0613
0.67	0.0001	0.8898	0.0001	0.1097	0.0000	0.9383	0.0000	0.0616

Table 21: Normalized confusion matrix for the label “chill” for the models on clusters

sum of the counts of the rest of the labels. On another note, the number of true positive and false positive are larger in $k = 100$ than in $k = 1000$, this is a consequence of the fact that the number of positive predictions is larger in the former than in the latter. Specifically, for $\tau = 0.5$, 4.051% of the predictions were positive for $k = 100$ and 1.711% were positive for $k = 1000$. This is a consequence of the fact that the number of samples that contain a given label for $k = 100$ is larger than the ones that contain it for $k = 1000$. See that we use the same number of tags for creating both sets of clusters, we filter them according to the filtered tags in Appendix A.3, hence the final number of clusters is less than or equal to 80 for both models and, on average, the model with $k = 100$ has more tags per cluster than the one with $k = 1000$. The mean of the number of tags per cluster is approximately 73 for $k = 100$ and 17 for $k = 1000$. Since there are more positive samples per cluster for $k = 100$, the CNN learns more features to characterize the clusters. This explains why the number of clusters and the percentage of positive predictions are inversely correlated. As a consequence, the TP and FP tend to increase as we decrease the number of clusters.

Finally, comparing the models with $k = 100$ and $k = 1000$, we may see that in the latter the TN rate increases consistently with respect to the former. On the other hand, the rate of FN increases slightly for the objective label “rock” and decreases for “chill”. This result is related to the fact that, in the former, the number of tags per cluster is greater than in the latter. The clusters contain at least one filtered tag, hence, we may expect that the tags that are included in the clusters for $k = 100$ but excluded from the ones with $k = 1000$ are the ones whose embeddings are further from the filtered tag, which we consider the most robust tag in the cluster. This implies that, effectively, the model with $k = 100$ is learning more features to recognise each cluster; however, the extra information that we are providing fits worse the tracks than the one available for $k = 1000$. This effect is magnified for subjective labels such as “chill”. Therefore, the number of negative predictions for $k = 100$ is less than the one for $k = 1000$. For the subjective tags, the accuracy is greater in the latter than in the former, which leads to a larger number of FN and a smaller amount of TN.

The discrepancy in the metrics is a consequence of the previously discussed divergences be-

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.4105	0.7953	0.3557	0.7584
Spectrogram	**0.4225**	**0.8009**	0.3661	0.7640

Table 22: Results of the model on clusters using the optimum set of hyper-parameters

tween models. It suffices to recall that the AUC PR increases as the precision and TPR increase and the AUC ROC increases as the TPR increases and the TNR decreases; where TPR $= \frac{\text{TP}}{\text{TP+FN}}$, FNR $= \frac{\text{FN}}{\text{TN+FP}}$ and precision $= \frac{\text{TP}}{\text{TP+FP}}$. We could consider that increasing k increases the accuracy of the classifier on the negative predictions and decreasing it improves the performance on the positive predictions. None of the options is objectively best; however, in music classification we may consider that a classifier that is accurate in assigning a genre to a song is more useful than one that stands out in predicting which genres do not belong to a track. Therefore, we will select Glove Wikipedia embeddings, $n = 100$ and $k = 100$ as the optimum set of hyper-parameters. The table containing all the clusters of the foregoing model can be found in the GitHub repository of the project[12].

Lastly, we have compared the performance of the model with the optimum set of hyper-parameters, hereinafter the model on clusters, to the one of the baseline model on tags. Comparing the results of Table (19) with the model on 80 filtered tags, Table (12), it could seem that the model on clusters outperforms the model on tags; however, to carry out a fair comparison, we must compute both models on the same number of tags.

In Table (22) we show the performance of the model on clusters with the optimum hyper-parameters and in Table (23) we present the results for the baseline model on the most popular tag of each cluster. See that we have reduced the number of tags of the latter model to 39. This implies that the clustering has filtered the tags according to its semantic information, removing the ones that contained similar information to other more popular tags. This duplicity of tags lead to classes that are characterised by similar or overlapping audio features, worsening the performance

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.4224	0.8549	0.3360	0.8172
Spectrogram	**0.4458**	**0.8653**	0.3479	0.8263

Table 23: Results of the baseline model on the most popular tag of each cluster

[12]`https://github.com/carlorop/Music_Tagger_on_Word_Embeddings`

Threshold	Model on clusters				Model on tags			
	TP	TN	FP	FN	TP	TN	FP	FN
0.33	0.2498	0.4821	0.1954	0.0725	0.2350	0.4956	0.1894	0.0798
0.50	0.1819	0.5765	0.1011	0.1403	0.1525	0.6037	0.0813	0.1623
0.67	0.0746	0.6507	0.0269	0.2475	0.0600	0.6653	0.0198	0.2548

Table 24: Normalized confusion matrix for the label “rock” for the model on tags and the model on clusters the optimum hyper-parameters

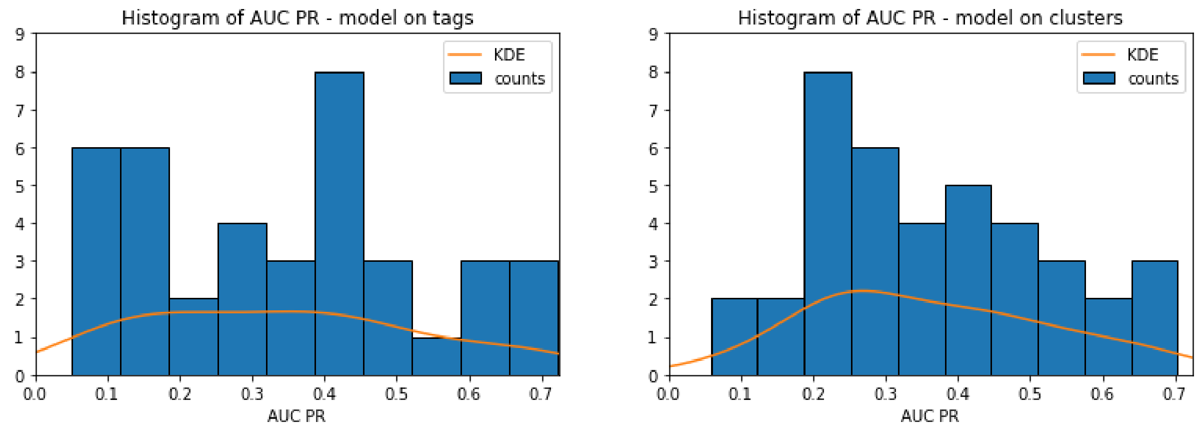
of the classifier. For this reason, the model on tags inferred from the clusters outperform the one on 80 tags computed on Section 6.2.

We may see that the model on tags outperform the model on clusters in the micro-averaging scheme but worsens the performance in the macro-averaging scheme. First of all, we must take into account that both results refer to the same model trained on different datasets. The dataset that uses tags as clusters is more robust, i.e., the labels suit best the tracks. The samples used to train the model on clusters include the ones used for the model on tags in addition to samples that contained labels that belong to any cluster without being their most popular tag, hence they are not the most robust ones. Accordingly, the metrics of the model on clusters are more penalised by the misclassification of the training samples. For this reason, we would expect to see an improvement in the micro-averaged metrics for the model on tags.

We may extend our analysis by inspecting the AUC PR per label, in Appendix A.4 we show the AUC PR for each label in both models. As we may see, for the objective tags, such as genres, the best performance is reached in the model on tags; however, for the most subjective tags, the best model is the one on clusters. In Table (24) and Table (25) we show the confusion matrix for an objective label, “rock”, and a subjective one, “chill”, respectively. We may see that the model on clusters consistently increases the rate of TP and FP due to the aforementioned reasons. It improves specially on the subjective tags, such as “chill” in Table (25). It can be explained taking

Threshold	Model on clusters				Model on tags			
	TP	TN	FP	FN	TP	TN	FP	FN
0.33	0.0134	0.8579	0.0324	0.0965	0.0021	0.9259	0.0056	0.0662
0.50	0.0019	0.8870	0.0031	0.1080	0.0001	0.9314	0.0001	0.0682
0.67	0.0001	0.8898	0.0001	0.1097	0.0000	0.9315	0.0000	0.0684

Table 25: Normalized confusion matrix for the label “chill” for the model on tags and the model on clusters using the optimum hyper-parameters



(a) Baseline model on tags (b) Model on clusters of word embeddings

Figure 13: Histograms of the AUC PR

into account that including more samples extends the range of features that characterise the label. This extension is much more pronounced for subjective tags than for objective tags, for instance, the cluster of "chill" includes similar concepts such as "sweet", "soft" or "cold" whilst the cluster of "rock" include rock subgenres. For this reason, the FP increases and the TN decreases for the subjective tags, increasing the FPR and hence lowering the AUC ROC.

In Figure (13) we show the histogram of the AUC PR for both models. In the model on tags, Figure (13a), we may see a larger density of counts for low AUC PR, these labels correspond to the subjective tags. In the model on clusters, Figure (13b), we may see how these counts are shifted toward higher values of AUC PR. In particular, out of the 39 classes, the model on embeddings outperforms the model on clusters on 22; however, these classes have, on average, fewer counts than the ones in which the model on tags achieves the best performance.

The metrics are useful tools to compare models; however, they cannot capture all their characteristics. In this case, including word embeddings in the model has lead to an overconfident classifier which, in general, commits more errors but also makes more correct positive predictions, especially, in the more subjective tags whose features cannot be captured by the model on tags. Choosing the model on tags or on clusters is a matter of the importance of the positive predictions on the classification problem.

For the sake of facilitating replication, we have uploaded the checkpoints for this model on clusters with optimum hyper-parameters[13] to the GitHub repository[14].

[13]See that we have not uploaded the checkpoints for all the models due to the limitation of the storage capacity, the checkpoints occupy more than 1 Gb for each model

[14]`https://github.com/carlorop/Music_Tagger_on_Word_Embeddings`

7.5 Model 3.4: Classification on genres

7.5.1 Setup

In the previous models we have proved that the tags and clusters representing musical genres led to better performances than the rest of the labels. This fact is not surprising, since we would expect that the musical patterns that distinguish genres are easier to discern from the raw spectrogram or waveform than the patterns that characterise concepts such as “happy”.

In the set of clusters, which is found in the repository[15], we may see that the clusters of genres contain several subgenres of the main genre, which is usually the most popular tag in the cluster. In this model we will train a classifier on clusters that represent genres, preprocessing the genres to avoid multiplicities, furthermore, we will compare it with the classifier on tags representing the main genre of each cluster.

For preprocessing the genres, we have computed the co-occurrence matrix of the clusters. We denote as n_{ij} the number of times in which clusters i and j appear in the same track. Equivalently, n_i represents the number of occurrences of cluster i. See that the most popular tags lead to a large number of co-occurrence between clusters, hence we shall define a normalized version of the co-occurrence matrix $(\tilde{C}_{ij})$ as follows:

$$\tilde{C}_{ij} = \frac{n_{ij}}{\sqrt{n_i n_j}}$$

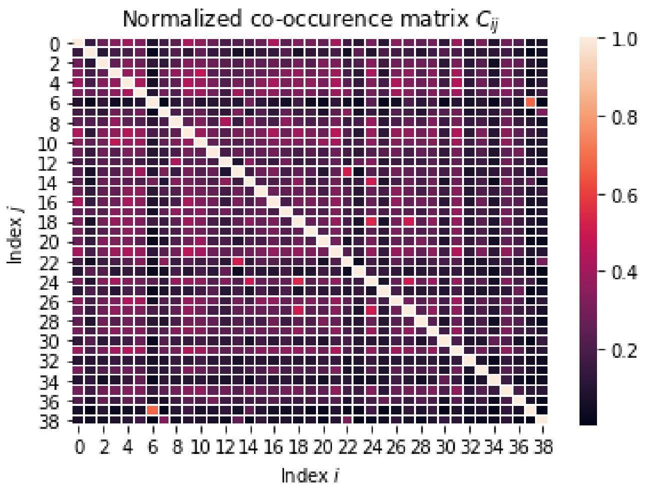


Figure 14: Heat map of the normalized co-occurrence matrix

[15]`https://github.com/carlorop/Music_Tagger_on_Word_Embeddings`

In Figure (14) we may see the heat map of the normalized co-occurrence matrix, as we may see the clusters are just slightly correlated, this is an indicator of the quality of the clustering. Quantitatively, $\tilde{C}_{ij} > 0.5$ just in 0.657% of the pairs of different clusters.

To get a representation as independent of the size of the cluster as possible, we will define the m-correlated clusters as follows. We will say that a pair of clusters is m-correlated, with $m \geq 1$, if the number of co-occurrences of the clusters is larger than the number of counts of the smaller cluster divided by m. Without lose of generality let us assume that $n_i > n_j$, given a pair of clusters (i,j), these clusters are m-correlated if $n_{ij} > n_j/m$. In other words, if the normalized co-occurrence matrix verifies:

$$\tilde{C}_{ij} = \frac{n_{ij}}{\sqrt{n_i n_j}} > \min\left(\sqrt{\frac{n_j}{n_i}}, \sqrt{\frac{n_i}{n_j}}\right)/m$$

We will make use of the above definition to merge the m-correlated of genres. Among the clusters representing genres, using $m = 3/2$, we obtain that the highly correlated clusters are the ones representing "metal" and "heavy metal". We could see this result in Figure (14), where these clusters were represented by labels 6 and 37 respectively, leading to the largest non-diagonal component of the co-occurrence matrix.

For the comparison with the baseline model we will consider just the tags that correspond to the most popular tag of each cluster of genres. Moreover, we will merge the tags "metal" and "heavy metal". In the model with clusters, we will use the clusters generated using the Glove Wikipedia algorithm with $n = 100$ and $k = 100$.

7.5.2 Results

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.6475	0.8861	0.6028	0.8601
Spectrogram	**0.6587**	**0.8901**	0.6275	0.8710

Table 26: Results of the model on clusters of genres

As we expected, the performance of the classifier on clusters of genres, Table (26), is greater than the one on clusters, Table (22). In Table (27) we may see a breakdown of the AUC PR for each genre for both models, regarding the model that uses all the clusters, the average AUC PR for the genres is 0.5593, which is less than the macro-averaged AUC PR for the model on clusters

Genre	soul	pop	metal	electronic (Ambient)	electronica (Trance)	blues
AUC PR on genres of Model 3.3	0.4193	0.4917	0.6176	0.5594	0.5361	0.4683
AUC PR on genres of Model 3.4	0.4581	0.5823	-	0.7570	0.6113	0.5746
AUC PR on the comparison model	0.5321	0.6003	-	0.6952	0.3686	0.5637
Genre	rock	jazz	indie	Hip-Hop	heavy metal	House
AUC PR on genres of Model 3.3	0.6409	0.6064	0.5469	0.7034	0.4430	0.6781
AUC PR on genres of Model 3.4	0.5167	0.7145	0.6689	0.5796	0.5228	0.7682
AUC PR on the comparison model	0.7441	0.7177	0.5681	0.7687	0.6128	0.4695

Table 27: Comparison of the performances of genres for Model 3.4, Model 3.3 and comparison model. The comparison model classifies only on the most popular tag of the clusters of Model 3.4. See that we have merged the clusters “metal” and “heavy metal” in the models on genres

of genres. To explain this discrepancy, we should recall that we filter the samples of the dataset to train just with the ones that contain a label that is included in the classes of the classifier. Since the classes of the model on genres are a subset of the ones of the model on clusters, the samples used to train the former are a subset of the ones used in the latter. On the other hand, we may assume that most of the tracks fit into a genre since there are 897 subgenres in the clusters. This implies that the training samples of the model on clusters that were excluded from the training of the model on genres actually suited some class of this model. Consequently, these samples are inducing errors in the classifier on clusters. If the track belongs to a genre but is not labelled as such then it could penalise the weights incorrectly in the training phase or lead to erroneous FP.

In Table (28) we show the performance of the comparison model, such model classifies on the most popular tag of the clusters of Model 3.4. In Table (27) we present the comparison of the foregoing model and Model 3.4, as we may see, it keeps the trends of the previous models. The performance has improved or does not vary in the most robust genres, such as “rock” or “pop”, this is caused by the fact that the addition of tags in the cluster increases extensively the number of features that characterize the label, including features that are less characteristic and hence the tracks that include them are more prone to misclassification. It is worth highlighting the case of the tags “electronic”, “electronica” and “House”; in these tags Model 3.4 outperforms the

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.6339	0.9023	0.5754	0.8802
Spectrogram	**0.6598**	**0.9093**	0.6049	0.8905

Table 28: Results of the model on tags using just the most popular tag of each cluster of genres

comparison model. The cluster represented by “electronic” encompasses different subgenres of electronic music, especially the ones related to experimental and ambient electronic music. On the other hand, the cluster whose most popular tag is “electronica” includes different subgenres of trance music, which consist of the combination of techno music with other kinds of genres such as pop. The cluster “House” comprised subgenres of House Music. Therefore, the aforementioned clusters were characterised by different features; however, the most popular tags of these clusters are identified by overlapping features, which justifies the decrease in the AUC PR for the comparison model.

8 Models on clusters based on co-occurrence

8.1 Model 4.1: Classification on subgenres via NPMI

8.1.1 Setup

As we saw in the previous models, some of the clusters correspond to musical genres. These clusters contain several subgenres of a main genre, which is usually the most popular tag in the cluster. The labels of the following models will be clusters of subgenres. The number of counts of the subgenres of any genre is significantly less than the total counts, therefore, we will be able to create clusters based on co-occurrence.

In this model, we will consider the tags as elements of a discrete space, hence, we will use the agglomerative hierarchical algorithm to perform the clustering. To that end, we are going to define a measure of similarity that captures the co-occurrences between the elements of the cluster of genres. We shall define this similarity from the concept of pointwise mutual information (PMI). The PMI is a measure of association commonly use in NLP which is derived from the co-occurrence matrix, it has been successfully to problems in the field of NLP, such as sentiment analysis [72] or review classification [73]. Let us denote by n_{ij} the number of co-occurrences of clusters i and j, we define n_i a the number of counts of cluster i and N as $N = \sum_{i,j} n_{ij}$, let $P(i,j) = {}^{n_{ij}}/_{N}$ and $P(i) = {}^{n_i}/_{N}$. We define the pointwise mutual information as [74]:

$$PMI(i,j) = \log_2\left(\frac{P(i,j)}{P(i)\,P(j)}\right) = \log_2\left(\frac{Nn_{ij}}{n_i n_j}\right) \tag{30}$$

See that $PMI(i,j)$ is defined only if $n_{ij} \neq 0$; however, we may normalize the PMI to avoid this drawback and prevent the bias induced by the imbalance of the dataset. The normalized pointwise mutual information (NPMI) is defined as follows [75]:

$$NPMI\,(i,j) = \frac{PMI\,(i,j)}{-\log_2(P(i,j))} = \frac{\log_2(P(i)P(j))}{\log_2(P(i,j))} - 1 \tag{31}$$

See that Equation (31) admits $n_{ij} = 0$, in this case $\frac{\log(P(i)P(j))}{\log P(i,j)} \to 0$, hence, in the case of completely uncorrelated clusters $NPMI\,(i,j) = -1$. On the other hand, if the co-occurrences of i and j appear at random we would expect $P(i,j) = P(i)P(j)$ leading to $NPMI\,(i,j) = 0$. Finally, if the clusters are completely correlated $n_{ij} = n_i = n_j$ then $NPMI\,(i,j) = 1$. Based on the foregoing reasoning, $NPMI\,(i,j) \in [-1,1]$. The measure of similarity must verify $d(i,i) = 0$ $\forall i$ and $d(i,j) = d(j,i)$ $\forall i,j$, the latter condition is verified for all functions of $NPMI\,(i,j)$, we may impose the former defining the similarity measure as:

$$d(i,j) = 1 - NPMI\,(i,j) \tag{32}$$

See that the similarity measure is not a distance function, since it does not verify the triangle inequality; however, this is not a requisite for clustering. To ensure that the similarity measure is suitable for the dataset we have computed the projection of the clusters using such a measure. We have projected the distance vectors of the dataset, in other words, for tag i, the component j of the distance vector is given by $(\mathbf{d}_i)_j = d(i,j)$. We would expect small values of the similarity between every tag and the main genre, leading to globular projections with the main genre in the centre. See that, since the space is discrete, we cannot use PCA for reducing the dimensionality, therefore, we will make use of the t-distributed stochastic neighbour embedding (t-SNE), which may reduce the dimensionality of the data from the distance matrix. In short, this technique computes a probability distribution for the dataset and aims to generate a similar distribution in the space with reduced dimensionality. In Figure (15) we may see the projection for the cluster of the genre rock, which verifies the foregoing conditions indicating that the similarity measure is suitable.

In this model, we have clustered the tags of each of the clusters of genres. We have used an agglomerative approach with $d(i,j)$, from Equation (32), as similarity function. We have computed the clustering using from 5 up to 20 clusters, selecting the set of clusters that maximize the silhouette score and removing all the singletons. We have compared the linkage criteria shown in Table (5), selecting the average linkage since it led to the best results.

We will use the union of the clusters of subgenres as the objective of the classifier. This union led to a set of 88 clusters, there are some clusters in which the tags have a clear relation, for instance in {electro funk , electro disco} or {latin pop , spanish pop, Rock pop, Pop Latino, pop

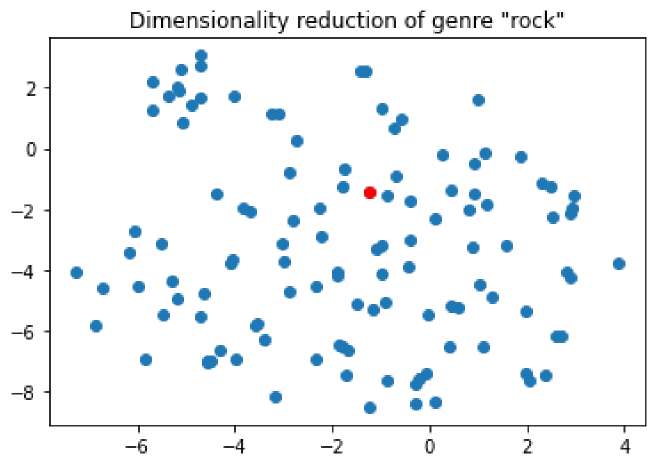


Figure 15: t-SNE projection of the distance matrix of the tags belonging to the cluster "rock"

espanol[16], spanish indie pop, rich spanish pop, female pop rock ocam, mexican pop}; however, there are other clusters with a spurious relation such as {christian pop, top pop songs, male pop rock ocam}.

8.1.2 Results

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.4408	0.9352	0.1751	0.8118
Spectrogram	**0.4594**	**0.8901**	0.1851	0.8261

Table 29: Results of the model on clusters based on NPMI

The results of the training, Table (29), stand out for the large discrepancy between the micro-averaging and the macro-averaging metrics; however, this fact is completely understandable. See that some clusters only contained a small number of tags with a little number of counts. For instance, out of the 88 clusters, 19 clusters contained just two tags (21.591%), this means that a large number of clusters are trained on just a few positive samples. We can measure this effect in the confusion matrix. Using the thresholds 0.33, 0.5 and 0.66 led to 36.363%, 43.181% and 56.818% of classes without any positive prediction respectively. This justifies the low performance of the macro-averaging scheme. See that a small rate of FP and a large rate of TN lead to a small FNR, which results in a large ROC if $\frac{\text{FN}}{\text{TP}}$ is small. It is noteworthy to mention that the

[16] "pop espanol" means Spanish pop in Spanish

large number of clusters that consisted of pairs of tags do not indicate a deficient clustering, in some cases this is the most convenient option, such as in the clusters $\{$dub techno, techno dub$\}$, $\{$visual kei , visual$\}$ or $\{$jam band, Jam Bands$\}$.

The labels with a small number of counts that worsened the macro-averaged metrics have very little influence on the micro-averaged scheme. The most popular clusters correspond to the most robust ones, and the classifier shows a good performance on these clusters.

8.2 Model 4.2: Classification on subgenres via tag embeddings

8.2.1 Setup

In this model, we will create embeddings of the classes of the classification problem. If there are N classes in the problem, we may associate class i with an N-dimensional vector whose components are 0 except for a 1 in position i. As in the one-hot encoding of the words of a vocabulary, such vectors do not contain any information about the relations of the classes. We may associate a tag embedding to each class, with dimension $d < N$, that captures the relations between different labels.

We shall associate the tag embeddings to tags that belong to the same cluster of genres. We will replicate the skip-gram model showed in Section 3.6.1 adapting it to our problem. The neural network will consist of three layers: the input layer, whose dimension is the size of the vocabulary, in our case, the number of tags in the cluster (N); the hidden layer, which transforms the input with size N in an output with dimension d; and the output layer, whose output has size N. The word embedding of tag i would correspond to the i-th row of the weight matrix associated with the hidden layer.

The input will be a one-hot encoding representing the tag, since the number of tags that belong to the same genre per track is reduced, we will not impose a window size, we will try to predict all the tags from the tracks but the input tag. For instance, if the combination of tags of a track is represented as $[0, 1, 1, 1]$, we would pass this vector three times to the neural network, it would correspond to the following samples $([0, 1, 0, 0], [0, 0, 1, 1])$, $([0, 0, 1, 0], [0, 1, 0, 1])$ and $([0, 0, 0, 1], [0, 1, 1, 0])$, where the first element is the input and the second one is the ground truth. See that these sets of labels are the tags that belong to the same cluster. A diagram of the neural network may be seen in Figure (16).

As a rule of thumb, the dimension of the word embeddings is usually the fourth root of the dimension of the vocabulary. In our case the vocabulary is the set of tags that belong to a cluster, whose order of magnitude is approximately 10^2, hence, we will take $\lfloor\sqrt{N}\rfloor$ as the dimension of the embeddings, instead of the fourth root.

We have trained the neural network using the binary cross-entropy as loss function and Adam as optimizer. We have used the softmax activation function, as dicussed in Section 3.6.1. The samples used for the training were inputs and outputs that only took into account the tags belonging to the same cluster, we trained different identical neural networks to obtain a set of embeddings per genre. Regarding the inspection of the embeddings, for instance, the cosine similarity between "Rock Espanol" (Spanish rock) and "rock español" is 0.937, whilst such a similarity between "Rock Espanol" and "surf rock" is 0.067. See that the surf rock was a subgenre of rock that originated in California in the sixties and whose greatest exponent was The Beach Boys, i.e., it is something completely unrelated to Spanish rock. These results indicate that the embeddings capture the semantic relations between the tags.

We have used tag embeddings to create clusters of subgenres that belong to the same main genre. In this case the space is continuous, hence we have used the modification of the k-means algorithm that we presented in Section 7.1. We have computed the clustering for k ranging from 5 to 20, selecting the cluster that maximises the silhouette score. Figure (17) depicts a diagram representing the silhouette function of the embeddings and the silhouette score for the clustering of the subgenres of rock.

The output of the model will be a vector in which each component represent one cluster of subgenres, joining the subclusters from all genres, which led to a vector with 110 components.

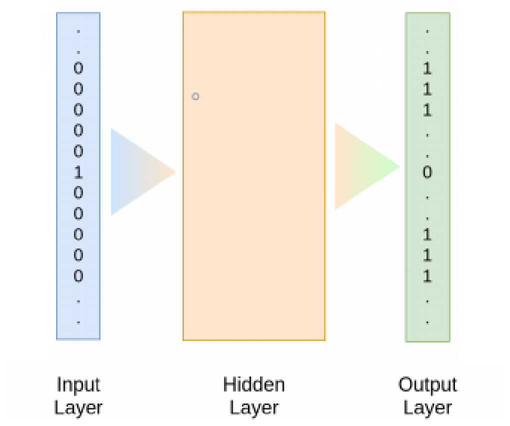


Figure 16: Skip-gram model. Figure taken from [76]

8.2.2 Results

Model	Micro-averaging		Macro-averaging	
	AUC PR	AUC ROC	AUC PR	AUC ROC
Waveform	0.3556	0.9234	0.1963	0.8592
Spectrogram	**0.3742**	**0.9300**	0.2054	0.8773

Table 30: Results of the model on clusters based on tag embeddings

It was expected that the model based on NPMI, Table (29), outperformed the model that used clustering based on tag embeddings, Table (30). See that the former computes the clustering directly from the co-occurrence of the labels; however, the latter computes embeddings based on the co-occurrence and uses these embeddings to create the clusters. For this reason, we would expect to find clusters with less similar tags in the model based on tag embeddings, therefore, the micro-averaged AUC PR has diminished in such a model.

As in the previous model, the results lead to a large discrepancy between the micro-averaged and macro-averaged schemes. In this case, the rate of classes without positive predictions for thresholds 0.33, 0.5 and 0.67 is 28.181%, 33.636% and 46.363% respectively, the percentage of clusters that contain two tags is 15.454%. We may infer that the classifier has received more positive predictions for the clusters will small number of counts, hence, its performance has

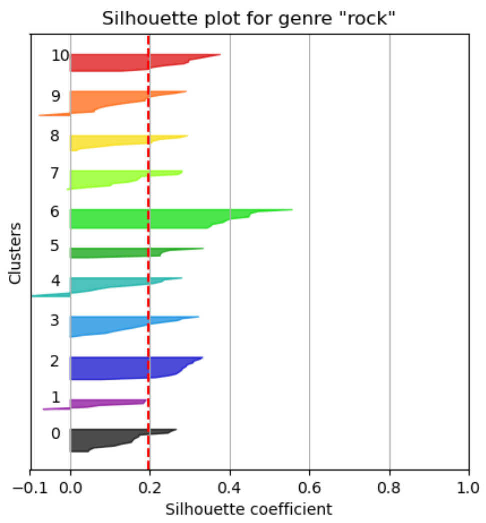


Figure 17: Silhouette function of the tags belonging to the cluster of rock. The broken line represents the silhouette score of the clustering and the coloured bars depict the silhouette function of the labels

increased in these clusters. These measurements justify the fact that the macro-averaged metrics improve in the model based on tag embeddings. See that the labels in which it improves are also the ones with a smaller number of counts, for this reason it does not outperform the model based on NPMI on the micro-averaged metric.

We may give a more rigorous argument in favour of the above hypothesis. We will prove the premise, which is equivalent to state that the dataset is more imbalanced with respect to the clustering in Model 4.1 than in Model 4.2. The standard way to measure class imbalance in multi-class problems is using the mean imbalance ratio (MeanIR) [77]. Let D be the dataset, $\mathcal{T}_i$ be the true labels of sample i and y_j denotes a class, with $j \in \{1, \cdots, n\}$ where n is the number of classes, we may define imbalance ratio as follows:

$$\mathrm{IR}(y_j) = \frac{\max_{y'} \left(\sum_{i=1}^{|D|} h\left(y', \mathcal{T}_i\right) \right)}{\sum_{i=1}^{|D|} h\left(y_j, \mathcal{T}_i\right)}, \quad h\left(y_j, \mathcal{T}_i\right) = \begin{cases} 1, & y_j \in \mathcal{T}_i \\ 0, & y_j \notin \mathcal{T}_i \end{cases}$$

The MeanIR is defined as:

$$\mathrm{MeanIR} = \frac{1}{n} \sum_{j=1}^{n} \mathrm{IR}(y_j)$$

See that if the classes are completely balanced then MeanIR $= 1$, the larger the MeanIR the more imbalanced the classes. For the clustering generated using NPMI, the MeanIR equals 71.2238, in the case of the classes generated by tag embeddings the MeanIR is 28.7327, therefore, we may confirm that the classes are imbalanced.

The other part of the hypothesis states that the tags with fewer counts lead to low performances, especially in the case of the clustering with NPMI. To verify this fact quantitatively, we have computed the Pearson correlation coefficient between the number of counts per label and the AUC PR. The value of the coefficient is $\rho = 0.7272$ and $\rho = 0.6708$ for the model with NPMI and tags embeddings respectively, with p-value $p < 0.001$ for both models. This result indicates that the AUC PR and the counts are correlated to a certain extent, being this correlation stronger in the clustering that used NPMI, just as we inferred.

9 Future Research

We have explored how to include word embeddings in the classification problem via clustering. However, we may make use of embeddings in different ways with the aim of improving the classification. We shall propose a different method for including word embeddings in this particular problem. This method similar to a technique used to improve classifiers on imbalanced datasets known as weighted loss. Weighted loss consists in defining the loss function as a variation of the cross-entropy or binary cross-entropy multiplying each term of the sum by a certain weight. There exist several variations of this technique in the literature [78, 79, 80, 81]. The loss function that we have used in the previous models is the binary cross-entropy defined in Equation (7), nonetheless, we could have used the following loss function per sample:

$$\mathcal{L}\left(\theta;\mathcal{M}\right)\left(\hat{y}_i\right) = -\frac{1}{n}\sum_{i=1}^{n}\left(t_i \log\left(\hat{y}_i\right) + \left(1-t_i\right)\log\left(1-\hat{y}_i\right)\left(1+\gamma(1-\max_{j\in\mathcal{T}}(\omega_i\cdot\omega_j))\right)\right) \tag{33}$$

Where $\mathcal{T}$ represents the set of true labels of the sample, $\hat{y}_i$ is the probability prediction for label i, γ is a hyper-parameter, ω_i represents a normalized word embedding and t_i equals 1 if i is one of the true labels of the sample and 0 otherwise. See that $\hat{y}_i \in (0,1)$, hence, for each iteration of the sum in which $t_i = 0$, the modification of the loss function is equivalent to adding a positive term proportional to $(1-\max_{j\in\mathcal{T}}(\omega_i\cdot\omega_j))$ to the binary cross-entropy. The justification of the max operator can be understood easily with an example. Let us assume that we pass a sample with the tags "rock" and "female voices". We expect a value for $\hat{y}_i$ close to 1 in the case of the tag "female voice", which lead to a large value of $\log(1-\hat{y}_i)$, in this case we would induce a small penalisation by adding the term $\log(1-\hat{y}_i)\gamma(1-\max_{j\in\mathcal{T}}(\omega_i\cdot\omega_j))$ since the cosine similarity between the embeddings of "female voice" and "female voices" is close to 1 independently of the similarity between "female voice" and "rock". See that if we had averaged the cosine similarity it would have led to random weights.

10 Conclusion

In this project we have presented several techniques to take advantage of the semantic information of the labels of a dataset generated by social tagging.

The first models presented were the baseline models on tags. The purpose of these models was to provide a suitable set of tags for classification. The performance of the baseline model on tags, Model 1, was worsened by the tags of the groups “Meaningless” and “Nationalities”. In Model 2 we measured the performance of the model removing these tags, verifying that the metrics improve.

One of the problems of datasets obtained by crowdsourcing with social tags is the multiplicity of labels that represent similar concepts. In Model 3.1 we presented a technique to maximize the retrieval of information from such datasets. It consists in clustering the word embeddings that represent the tags. In this way, the classes of the classification become groups of tags that carry similar semantic information. In Model 3.2 and Model 3.3 we perform a grid search and a hyper-parameter tuning to obtain the optimum set of hyper-parameters.

We have compared the models on clusters with the models on tags classifying on the most popular tag of each cluster. We may expect that, in some cases, such tags roughly encompass the musical features that characterise each cluster. However, see that a like-for-like comparison of both models is not feasible since the classes are different. In comparison with the baseline model on tags, the model on clusters learns a greater number of musical features to characterise each label, leading to a larger rate of positive predictions. This fact results in an increase of the true positive predictions but also of the false positives, for this reason, the performance metrics tend to decrease for the model on clusters when we use the same number of labels. We have seen that the model on clusters outperforms the one on tags for the subjective labels. This fact makes the model on clusters suitable for tasks that imply recognising subjective concepts such as emotion recognition, for instance, it has lead to a large increase in the correct predictions of the labels “melancholy” and “sad”, therefore, it has been able to distinguish both concepts. On the other hand, if we classify on tags we may reach larger performances on robust labels such as genres. In general, neither of the two models is better than the other, its suitability depends on the problem and the importance of the TP compared to the FP.

The performance attained by the classifier on tags using just the most popular tag of each cluster improved the one reached by the same model on the filtered tags. Consequently, the process of creating clusters of word embeddings can be used to filter automatically the tags based on the semantic information. This set of filtered tags led to larger performances than the one created filtering the unsuitable tags by hand since the features that characterise the labels of the former set do not overlap.

Lastly, we have extended the idea of classifying on clusters to the classification of subgenres. In Model 5.1 we perform the clustering based on a measure of similarity derived from the normalised pointwise mutual information (NPMI), which quantifies the co-occurrence between labels. In Model 5.2 we have created embeddings that encode the co-occurrence of the tags, using these embeddings for clustering the labels. The model using embeddings led to an improvement of the average performance per cluster but worsened the total performance. The clustering of classes, based on NPMI or embeddings, can be used to reduce the dimensionality of the output of the classification in multi-label problems.

Appendices

A Data inspections

A.1 100 most popular tags

Tag	Counts	Tag	Counts	Tag	Counts	Tag	Counts	Tag	Counts
rock	101,071	indie rock	24,619	Favourites	17,722	loved	12,483	cover	10,694
pop	69,159	Mellow	24,356	female vocalist	17,328	sad	12,425	techno	10,679
alternative	55,777	electronica	24,087	guitar	17,302	House	12,404	reggae	10,610
indie	48,175	80s	23,492	Hip-Hop	16,712	favorite songs	12,392	heard on Pandora	10,470
electronic	46,270	folk	23,492	70s	16,075	happy	12,291	relax	10,205
female vocalists	42,565	british	23,033	party	15,320	punk rock	12,251	new wave	10,202
favorites	39,921	90s	23,018	country	15,197	piano	12,237	relaxing	9,751
Love	34,901	chill	22,746	easy listening	15,076	psychedelic	12,057	upbeat	9,501
dance	33,618	american	22,694	sexy	14,706	hip hop	12,049	good	9,475
00s	31,432	instrumental	21,837	catchy	14,490	male vocalist	11,921	romantic	9,327
alternative rock	30,334	punk	21,203	funk	14,189	classic	11,913	epic	9,279
jazz	30,152	oldies	20,979	favourite	14,167	pop rock	11,863	Ballad	9,250
beautiful	29,421	seen live	20,705	electro	14,041	downtempo	11,466	melancholic	9,038
singer-songwriter	27,910	blues	20,474	heavy metal	13,830	trance	11,434	death metal	9,014
metal	27,430	hard rock	20,241	Progressive rock	13,406	melancholy	11,398	summer	8,820
chillout	27,276	cool	19,581	60s	13,316	female	11,374	USA	8,725
male vocalists	27,269	Favorite	18,864	fun	13,202	amazing	11,270	2000s	8,671
Awesome	26,248	ambient	17,982	rnb	13192	hardcore	11,223	Favourite Songs	8,661
classic rock	25,771	acoustic	17,889	indie pop	13026	rap	11,157	emo	8,622
soul	24,702	experimental	17,764	Soundtrack	12,899	lounge	11,113	UK	8616

Table 31: 100 most popular tags from the last.fm dataset

A.2 Tags breakdown by group

Group	Tags
Genres	Rock, pop, alternative, indie, electronic, dance, alternative rock, jazz, metal, chillout, classic rock, soul, indie rock, electronica, folk, punk, blues, hard rock, experimental, hip-Hop, country, funk, electro, heavy metal, Progressive rock, rnb, indie pop, house, punk rock, psychedelic, hip hop, classic, pop rock, downtempo, trance, hardcore, rap, techno, reggae, new wave, Ballad, death metal, emo
Instruments	guitar, piano
Dates	00s, 80s, 90s, oldies, 70s, 60s, 2000s
Voices	female vocalists, male vocalists, female vocalist, male vocalist, female
Emotions	Love, beautiful, mellow, chill, party, easy listening, fun, loved, sad, happy, melancholy, relax, relaxing, upbeat, romantic, melancholic
Nationalities	british, american, USA, UK
Meaningless	Favorites, awesome, seen live, cool, Favorite, Favourites, sexy, catchy, favourite, Soundtrack, favorite songs, amazing, heard on Pandora, good, epic, Favourite Songs
Miscellaneous	singer-songwriter, instrumental, ambient, acoustic, lounge, cover, summer

Table 32: 100 most popular tags from the last.fm dataset classified by groups

A.3 Filtered tags

Tag	Counts	Tag	Counts	Tag	Counts	Tag	Counts	Tag	Counts
rock	101,071	classic rock	25,771	experimental	17,764	loved	12,483	cover	10,694
pop	69,159	soul	24,702	female vocalist	17,328	sad	12,425	techno	10,679
alternative	55,777	indie rock	24,619	guitar	17,302	House	12,404	new wave	10,202
indie	48,175	Mellow	24,356	Hip-Hop	16,712	happy	12,291	reggae	10,610
electronic	46,270	electronica	24,087	70s	16,075	punk rock	12,251	relax	10,205
female vocalists	42,565	80s	23,492	party	15,320	piano	12,237	new wave	10,202
Love	34,901	folk	23,492	country	15,197	psychedelic	12,057	relaxing	9,751
dance	33,618	90s	23,018	easy listening	15,076	hip hop	12,049	hardcore	11,223
00s	31,432	chill	22,746	funk	14189	male vocalist	11,921		
alternative rock	30,334	instrumental	21,837	electro	14,041	classic	11,91		
jazz	30,152	punk	21,203	heavy metal	13,830	pop rock	11,863		
beautiful	29,421	oldies	20,979	Progressive rock	13,406	downtempo	11,466		
singer-songwriter	27,910	blues	20,474	60s	13,316	trance	11,434		
metal	27,430	hard rock	20,241	fun	13,202	melancholy	11,398		
chillout	27,276	acoustic	17,889	rnb	13,192	female	11,374		
male vocalists	27,269	ambient	17,982	indie pop	13,026	rap	11,157		

Table 33: Set of tags remaining after filtering the meaningless tags and the ones corresponding to nationalities from the 100 most popular tags in the last.fm dataset

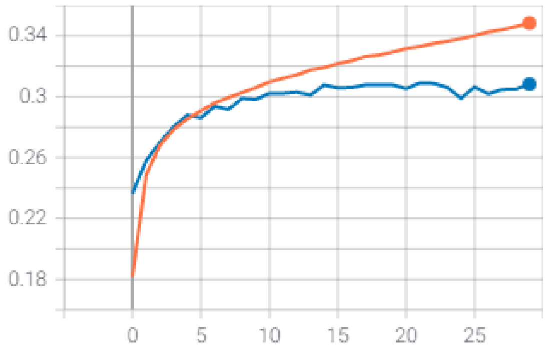
A.4 Comparison of the AUC PR

Label	Cluster		Tag		Label	Cluster		Tag	
	AUC PR	counts	AUC PR	counts		AUC PR	counts	AUC PR	counts
sad	0.1940	68,074	0.1171	12,425	singer-songwriter	0.2742	47,146	0.2835	27,910
soul	0.4196	41,759	0.4506	24,702	Love	0.2346	92,274	0.1726	34,901
acoustic	0.2799	77,141	0.2790	17,889	electronica	0.5361	96,969	0.3299	24,087
pop	0.4916	137,281	0.4922	69,159	blues	0.4683	53,636	0.5158	20,474
Mellow	0.2536	84,701	0.2124	24,356	rock	0.6409	214,580	0.6349	101,071
00s	0.4209	128,537	0.1774	31,432	jazz	0.6065	82,695	0.6375	30,152
metal	0.6176	89,241	0.6823	27,430	chill	0.2103	60,770	0.1784	22,746
dance	0.3594	50,694	0.4316	33,618	indie	0.5469	151.879	0.4710	48,175
80s	0.3963	109,172	0.3465	23,492	folk	0.3487	67.330	0.3948	23,492
beautiful	0.2481	100,655	0.2133	29,421	seen live	0.2375	96,200	0.1399	20,705
female vocalists	0.4786	147,128	0.5325	42,565	reggae	0.4508	53,631	0.7222	10,610
instrumental	0.3010	62,324	0.3945	21,837	easy listening	0.2387	108,842	0.1479	15,076
classic rock	0.26342	59,163	0.2858	25,771	cover	0.0587	28,011	0.0533	10,694
electronic	0.5594	115,726	0.6189	46,270	romantic	0.1044	29,260	0.0942	9,327
hard rock	0.3761	106,899	0.3936	20,241	Hip-Hop	0.7034	64,603	0.7000	16,712
lounge	0.1730	57,406	0.1770	11,113	american	0.3389	107411	0.1006	22,694
melancholy	0.2419	64,965	0.1143	11,398	piano	0.2698	40,515	0.2921	12,237
oldies	0.2267	61,111	0.3197	20,979	heavy metal	0.6781	7,1290	0.4470	13,830
alternative	0.4134	107,482	0.3972	55,777	House	0.4430	32,380	0.4064	12,404
summer	0.1470	47,607	0.0509	8,820					

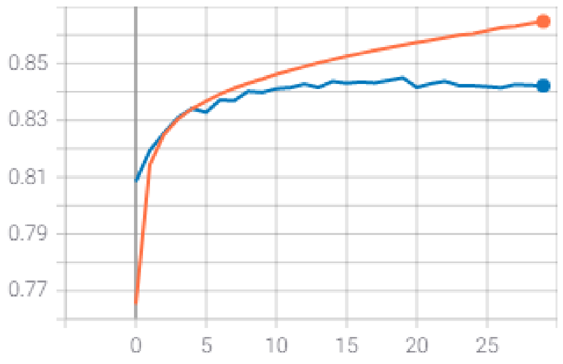
Table 34: Comparison of the per label AUC PR of Model 3.3 using clusters and tags. See that, in most cases, the most popular tag in the cluster is a filtered tag

B Training curves

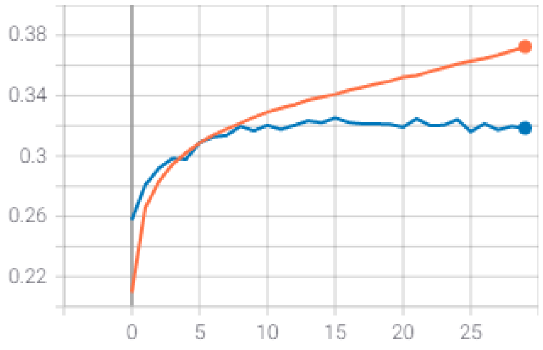
Model 1: Micro-averaging



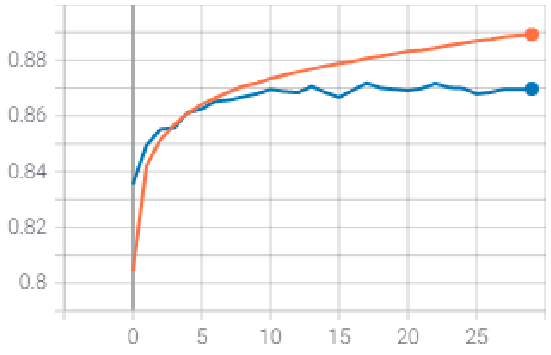
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



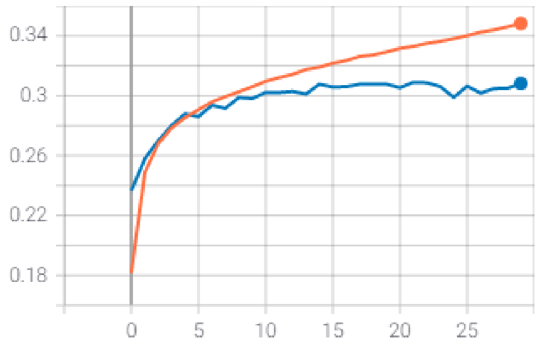
(c) AUC PR vs epoch spectrogram



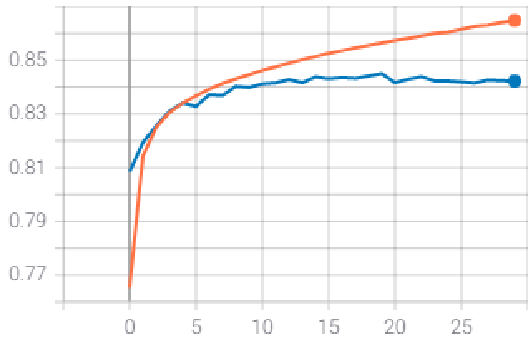
(d) AUC ROC vs epoch spectrogram

Figure 18: Micro-averaged metrics during training process of Model 1

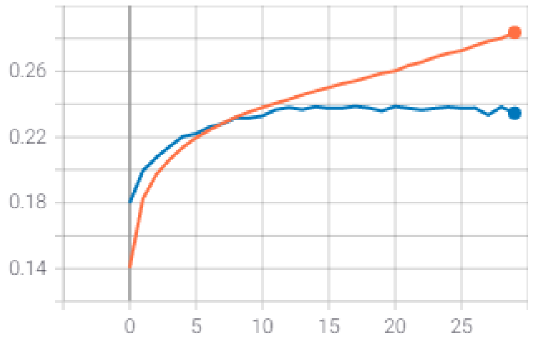
Model 1: Macro-averaging



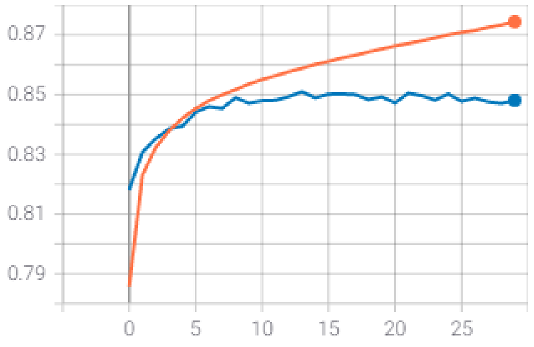
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



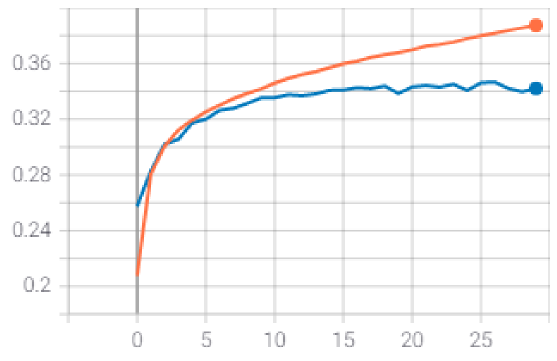
(c) AUC PR vs epoch spectrogram



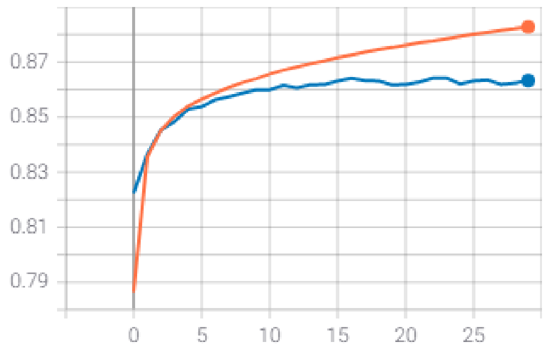
(d) AUC ROC vs epoch spectrogram

Figure 19: Macro-averaged metrics during training process of Model 1

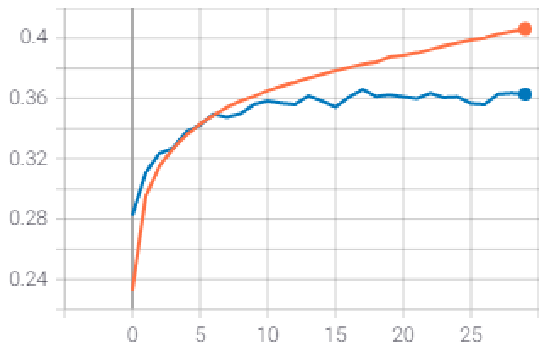
Model 2: Micro-averaging



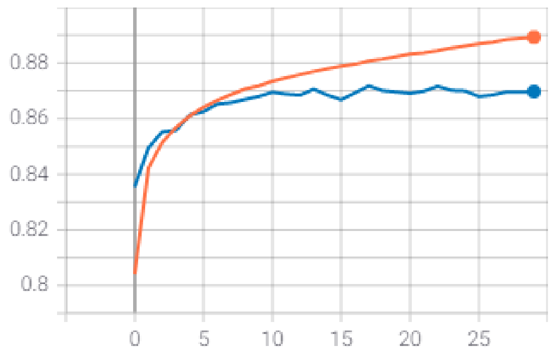
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



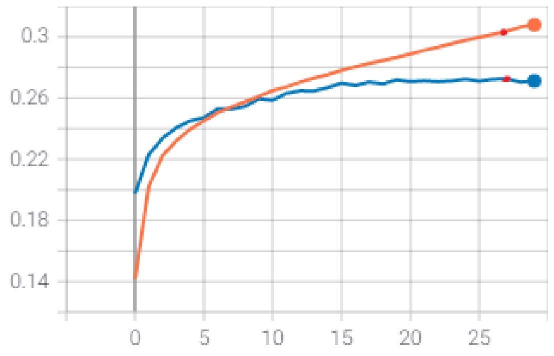
(c) AUC PR vs epoch spectrogram



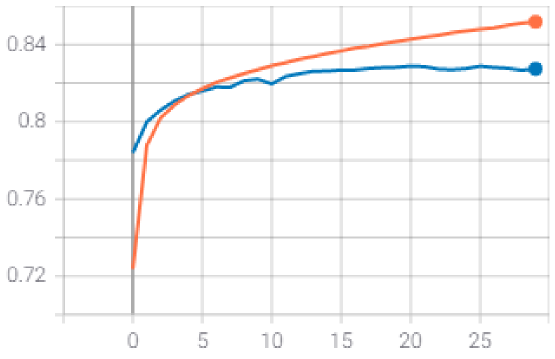
(d) AUC ROC vs epoch spectrogram

Figure 20: Micro-averaged metrics during training process of Model 2

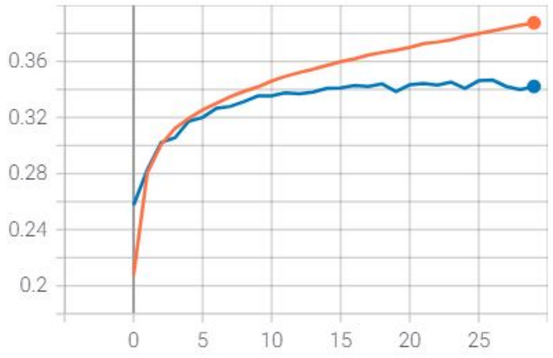
Model 2: Macro-averaging



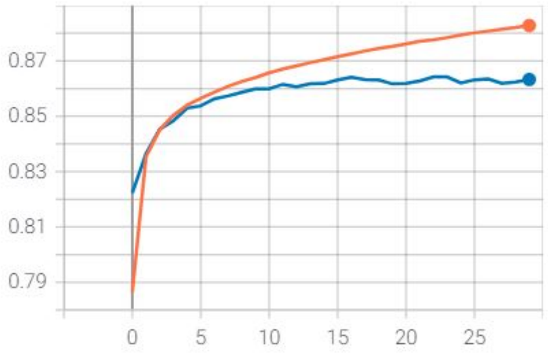
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



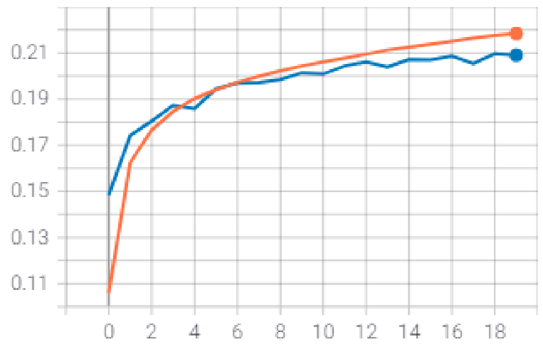
(c) AUC PR vs epoch spectrogram



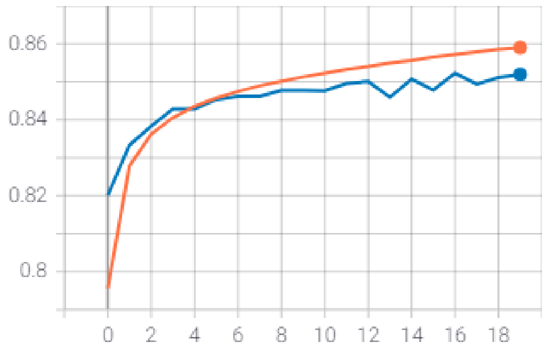
(d) AUC ROC vs epoch spectrogram

Figure 21: Macro-averaged metrics during training process of Model 2

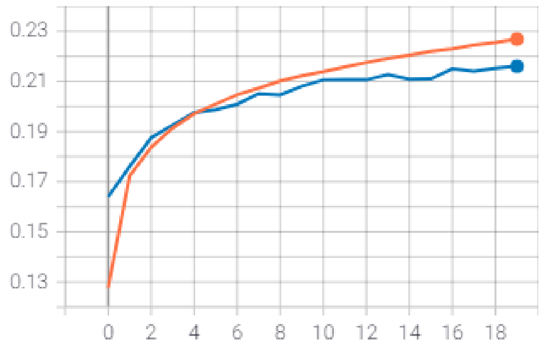
Model 3.1: Micro-averaging



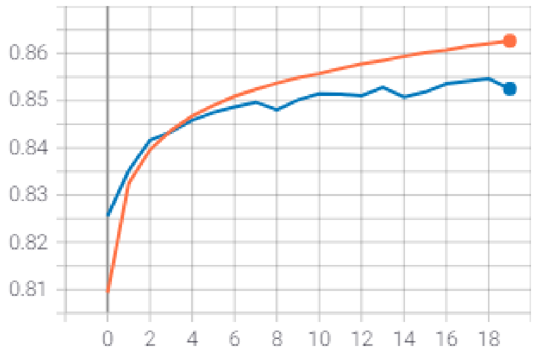
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



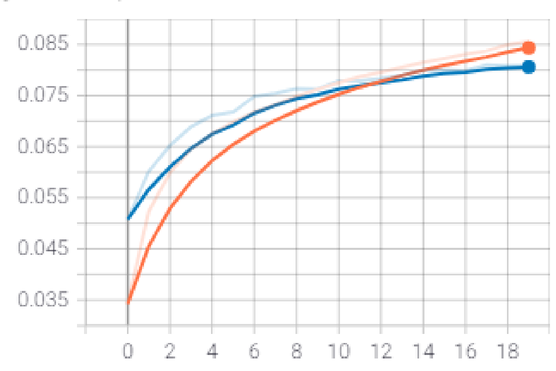
(c) AUC PR vs epoch spectrogram



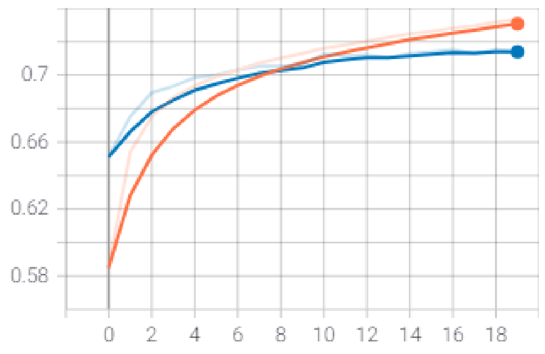
(d) AUC ROC vs epoch spectrogram

Figure 22: Micro-averaged metrics during training process of Model 3.1

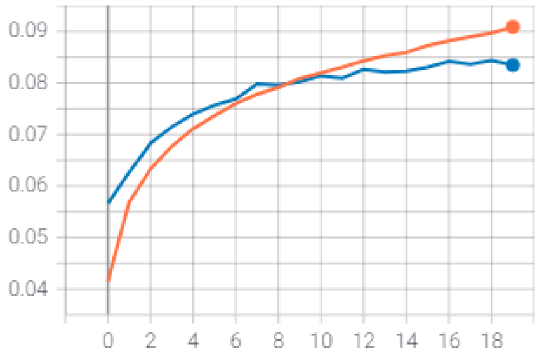
Model 3.1: Macro-averaging



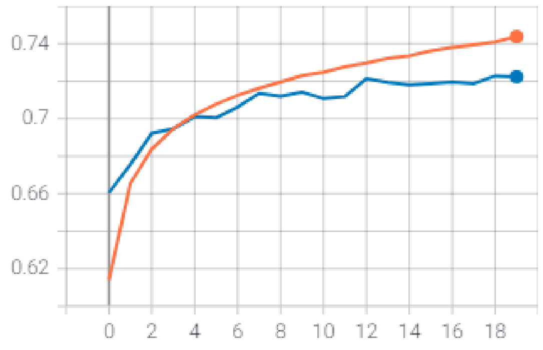
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



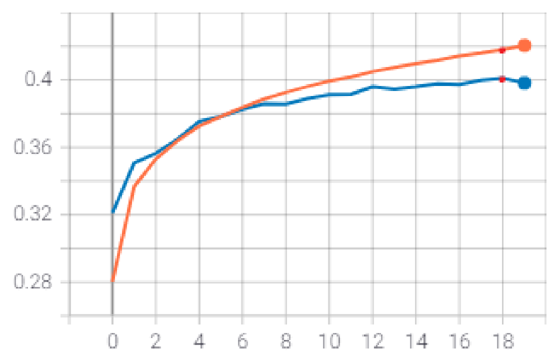
(c) AUC PR vs epoch spectrogram



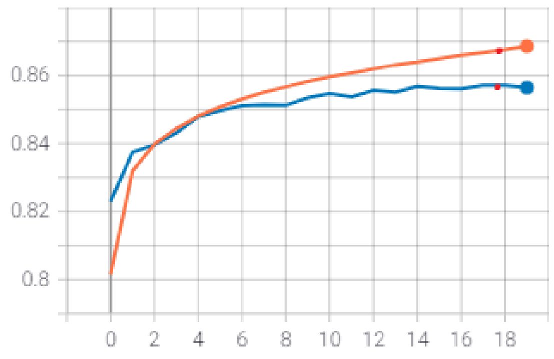
(d) AUC ROC vs epoch spectrogram

Figure 23: Macro-averaged metrics during training process of Model 3.1

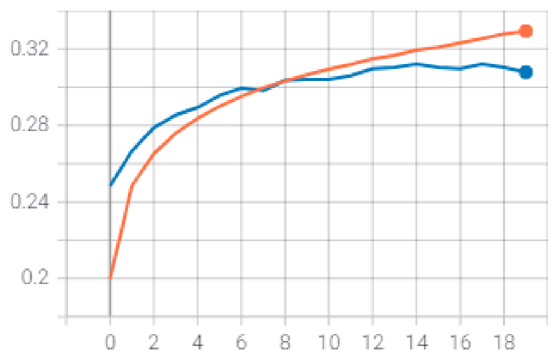
Model 3.2: Glove Wikipedia, $n = 100$, $k = 500$



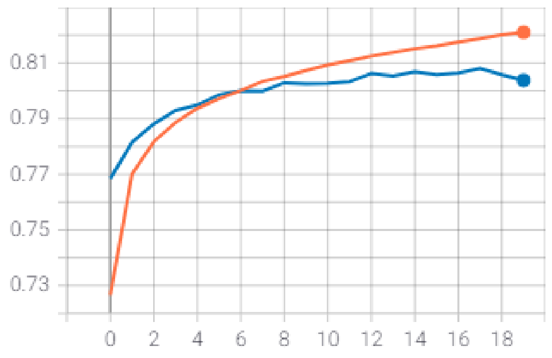
(a) Micro-averaged AUC PR vs epoch



(b) Micro-averaged AUC ROC vs epoch



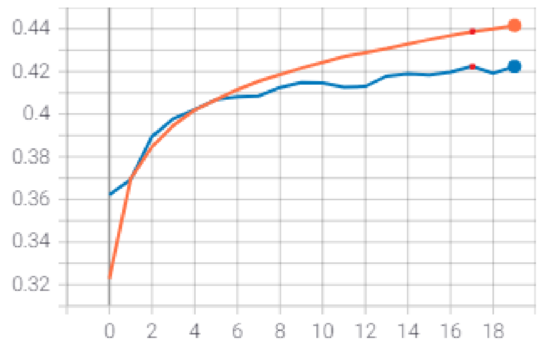
(c) Macro-averaged AUC PR vs epoch



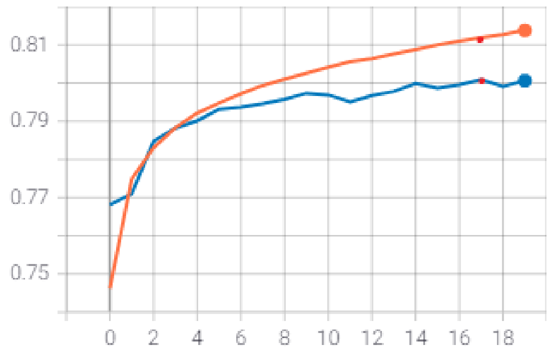
(d) Macro-averaged AUC ROC vs epoch spectrogram

Figure 24: Training metrics of Model 3.2 for embeddings generated by Glove Wikipedia with $n = 100$ and $k = 500$ using spectrogram inputs

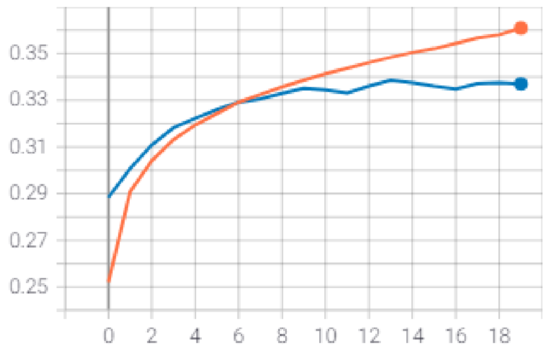
Model 3.3: Glove Wikipedia, $n = 100$, $k = 100$



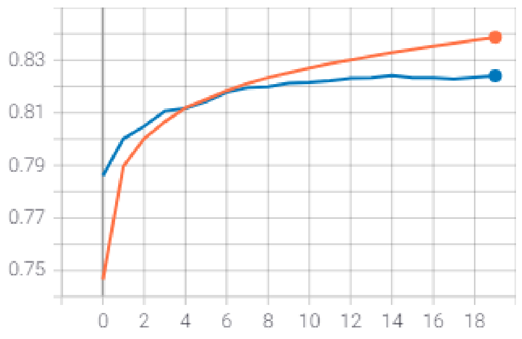
(a) Micro-averaged AUC PR vs epoch



(b) Micro-averaged AUC ROC vs epoch



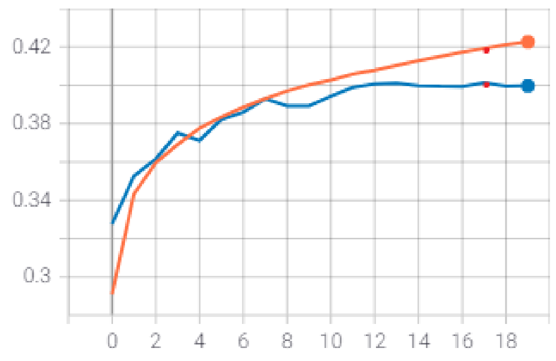
(c) Macro-averaged AUC PR vs epoch



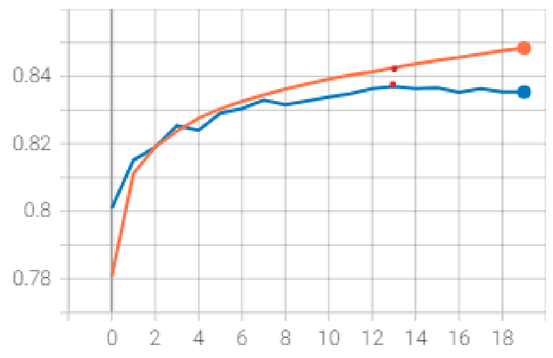
(d) Macro-averaged AUC ROC vs epoch spectrogram

Figure 25: Training metrics of Model 3.3 for embeddings generated by Glove Wikipedia with $n = 100$ and $k = 100$ using spectrogram inputs

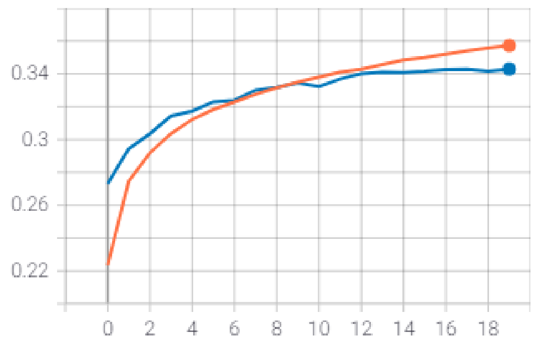
Model 3.3: Glove Wikipedia, $n = 100$, $k = 250$



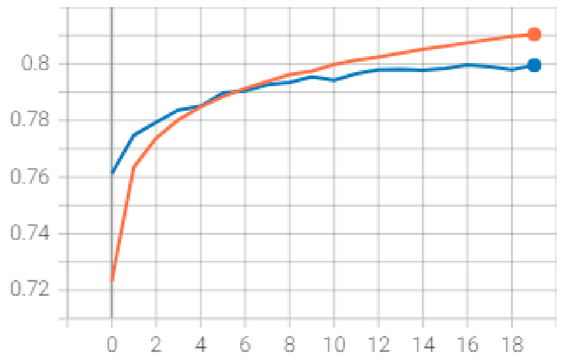
(a) Micro-averaged AUC PR vs epoch



(b) Micro-averaged AUC ROC vs epoch



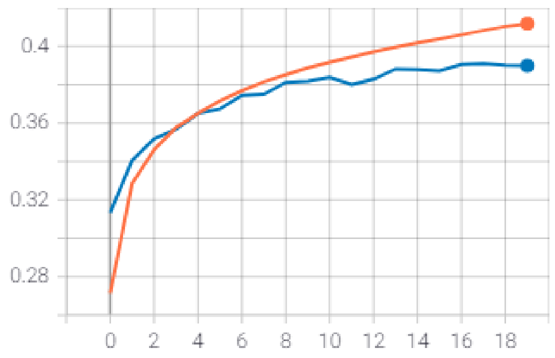
(c) Macro-averaged AUC PR vs epoch



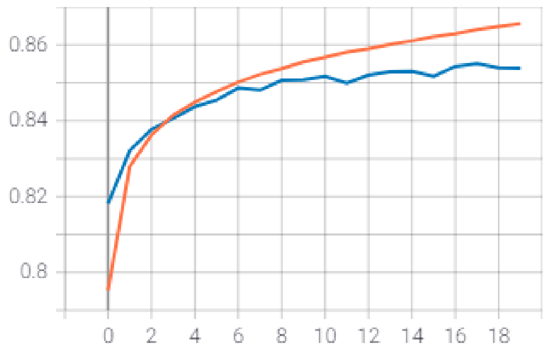
(d) Macro-averaged AUC ROC vs epoch spectrogram

Figure 26: Training metrics of Model 3.3 for embeddings generated by Glove Wikipedia with $n = 100$ and $k = 250$ using spectrogram inputs

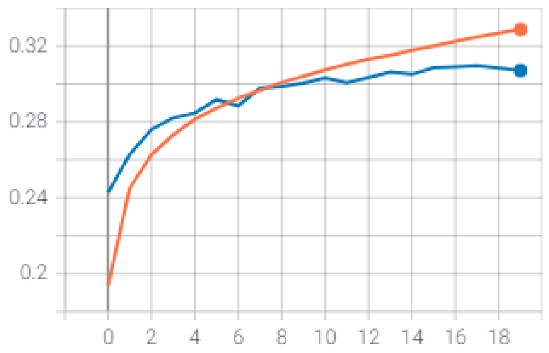
Model 3.3: Glove Wikipedia, $n = 100$, $k = 750$



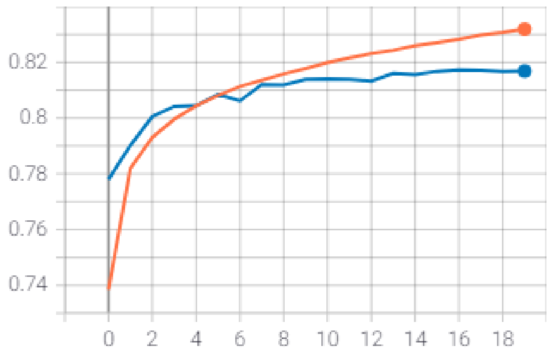
(a) Micro-averaged AUC PR vs epoch



(b) Micro-averaged AUC ROC vs epoch



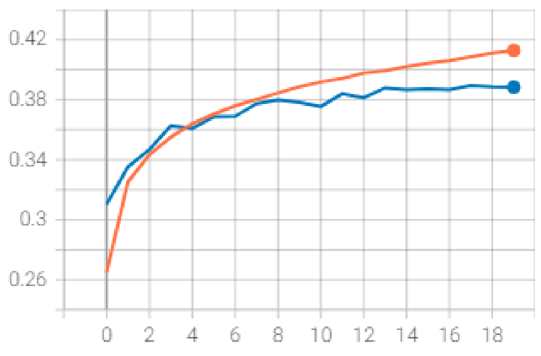
(c) Macro-averaged AUC PR vs epoch



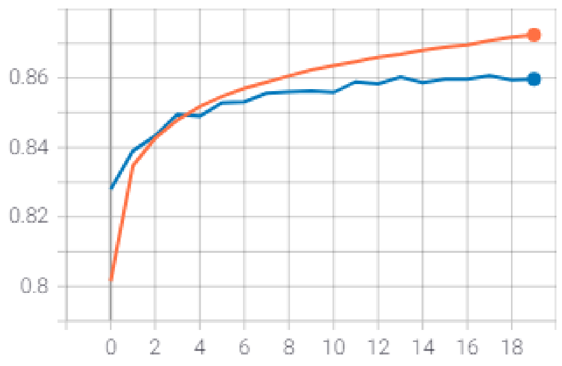
(d) Macro-averaged AUC ROC vs epoch spectrogram

Figure 27: Training metrics of Model 3.3 for embeddings generated by Glove Wikipedia with $n = 100$ and $k = 750$ using spectrogram inputs

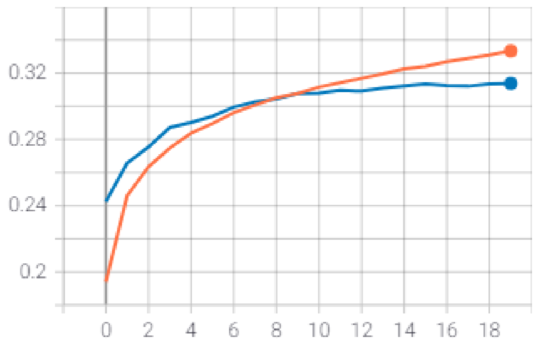
Model 3.3: Glove Wikipedia, $n = 100$, $k = 1,000$



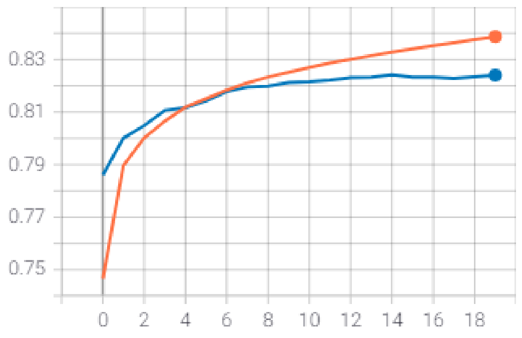
(a) Micro-averaged AUC PR vs epoch



(b) Micro-averaged AUC ROC vs epoch



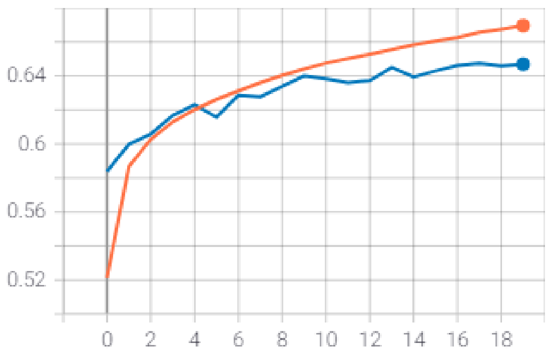
(c) Macro-averaged AUC PR vs epoch



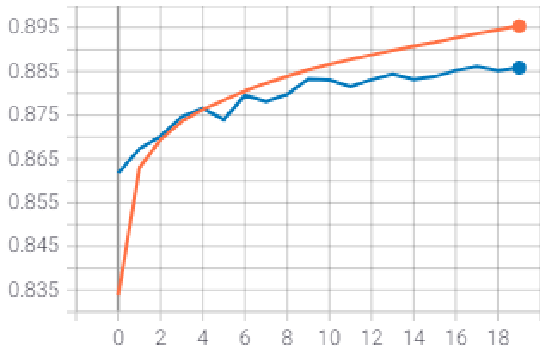
(d) Macro-averaged AUC ROC vs epoch spectrogram

Figure 28: Training metrics of Model 3.3 for embeddings generated by Glove Wikipedia with $n = 100$ and $k = 1,000$ using spectrogram inputs

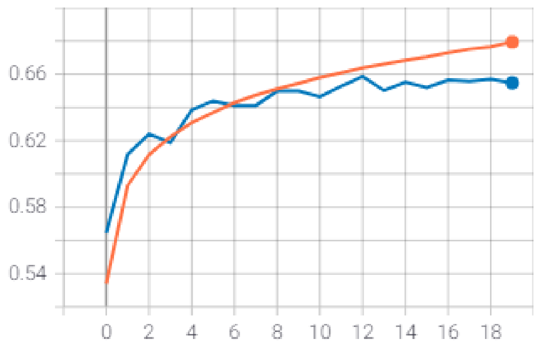
Model 3.4: Micro-averaging



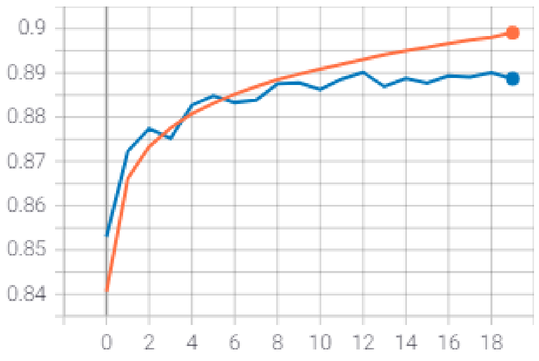
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



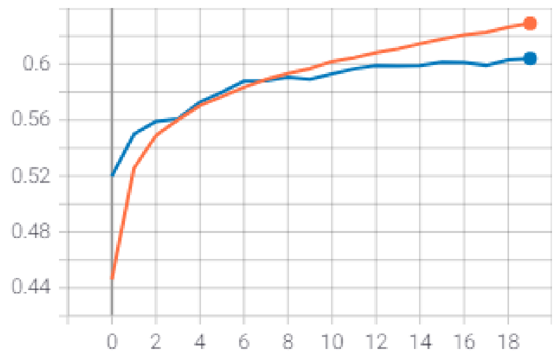
(c) AUC PR vs epoch spectrogram



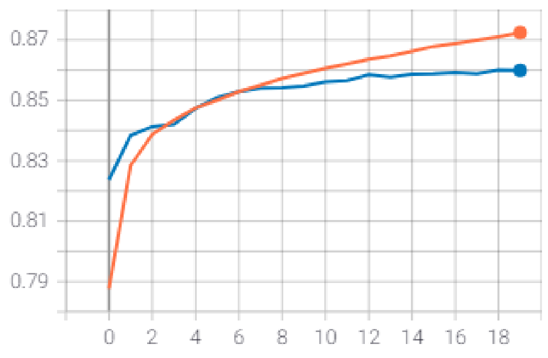
(d) AUC ROC vs epoch spectrogram

Figure 29: Micro-averaged metrics during training process of Model 3.4

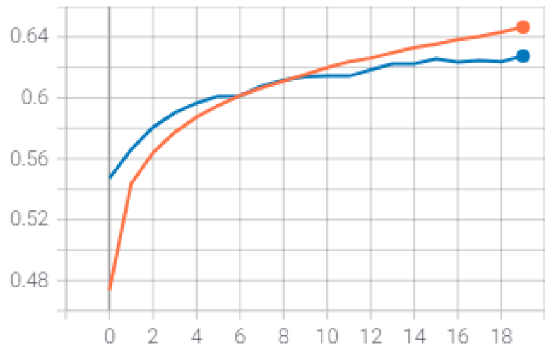
Model 3.4: Macro-averaging



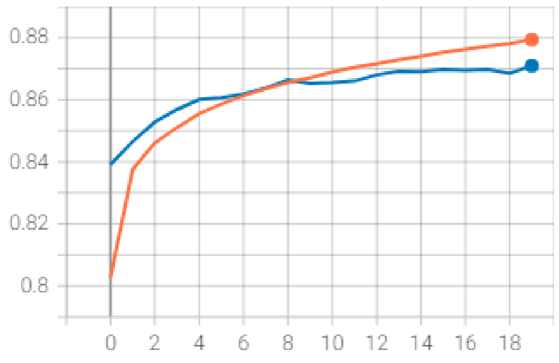
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



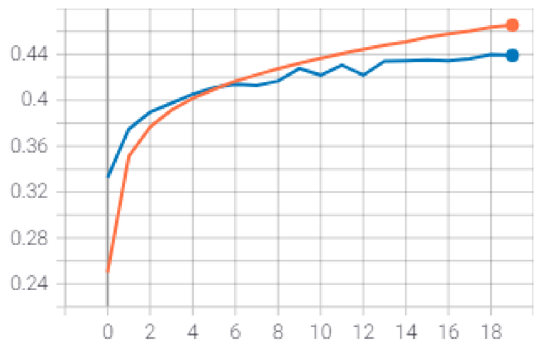
(c) AUC PR vs epoch spectrogram



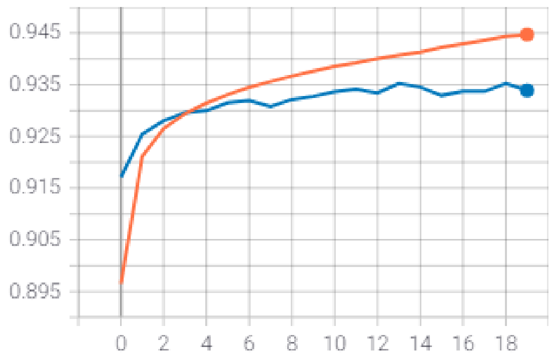
(d) AUC ROC vs epoch spectrogram

Figure 30: Macro-averaged metrics during training process of Model 3.4

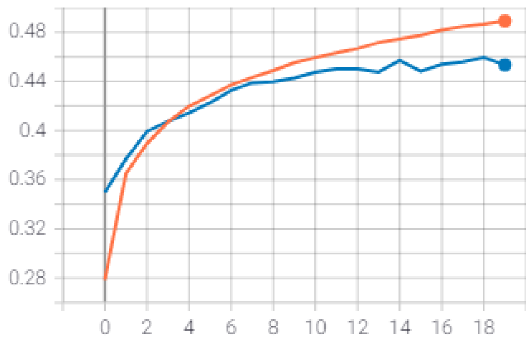
Model 4.1: Micro-averaging



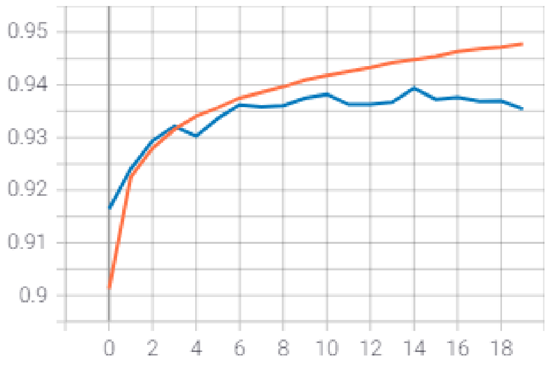
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



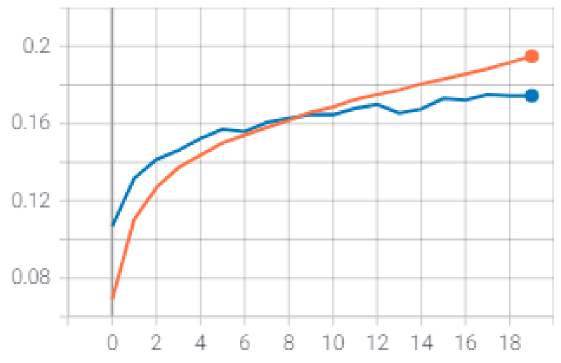
(c) AUC PR vs epoch spectrogram



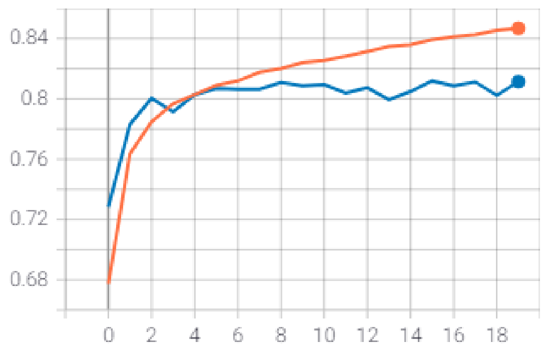
(d) AUC ROC vs epoch spectrogram

Figure 31: Micro-averaged metrics during training process of Model 4.1

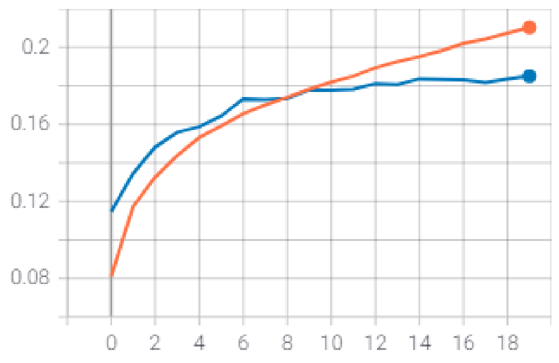
Model 4.1: Macro-averaging



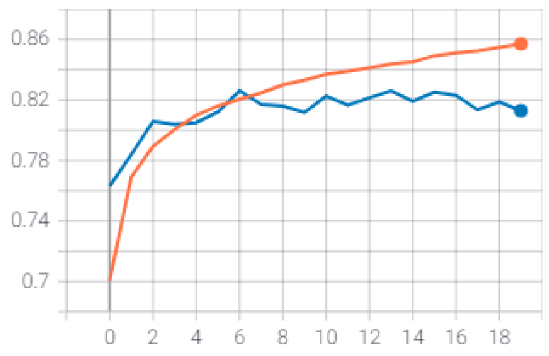
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



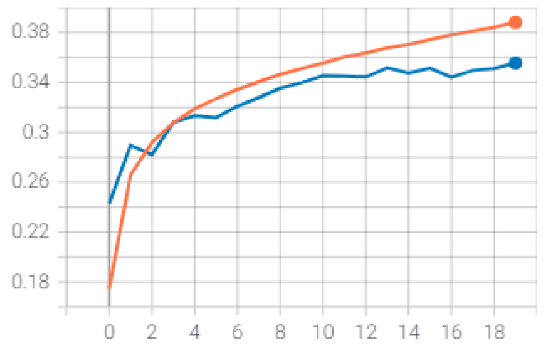
(c) AUC PR vs epoch spectrogram



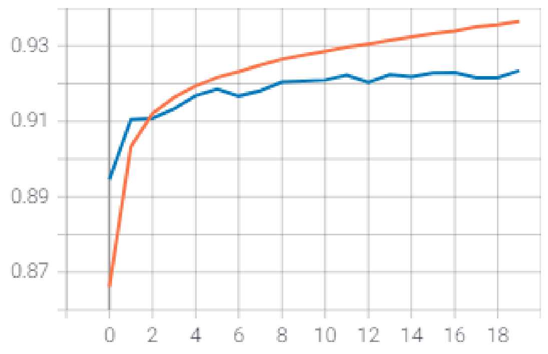
(d) AUC ROC vs epoch spectrogram

Figure 32: Macro-averaged metrics during training process of Model 4.1

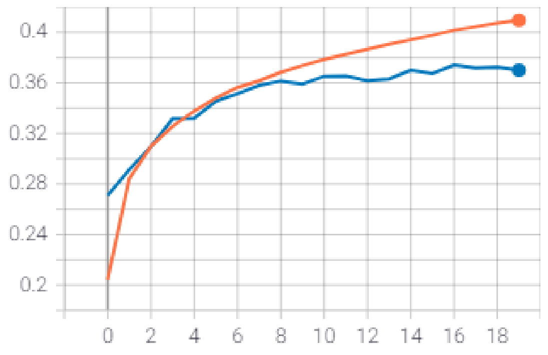
Model 4.2: Micro-averaging



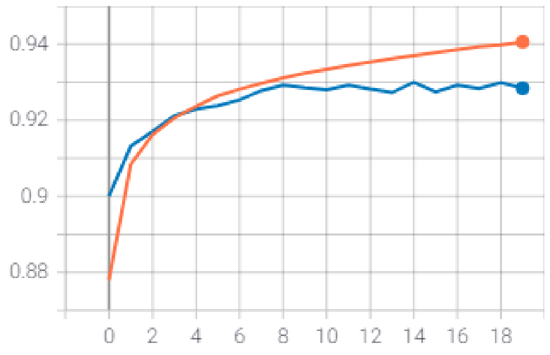
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



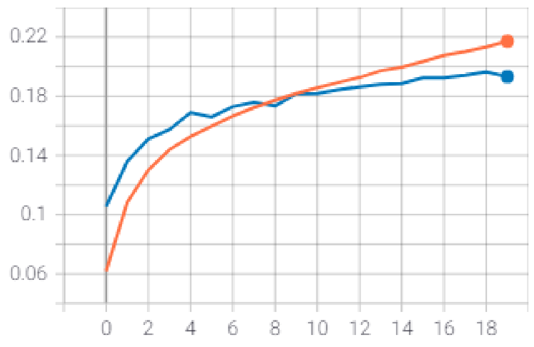
(c) AUC PR vs epoch spectrogram



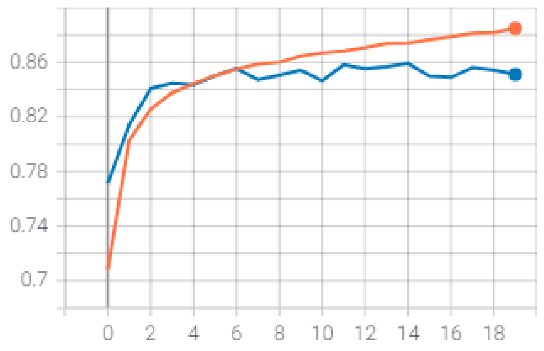
(d) AUC ROC vs epoch spectrogram

Figure 33: Micro-averaged metrics during training process of Model 4.2

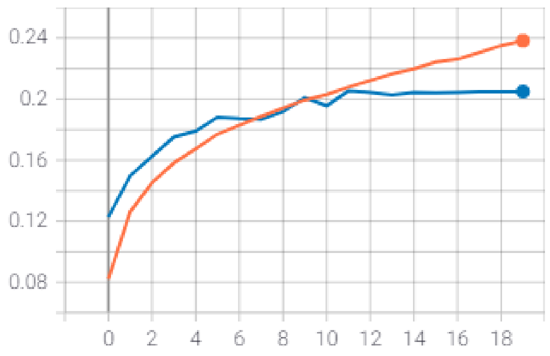
Model 4.2: Macro-averaging



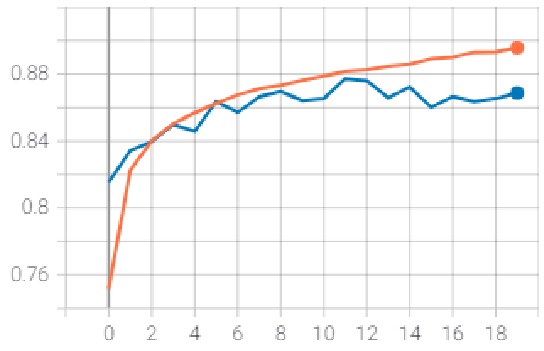
(a) AUC PR vs epoch waveform



(b) AUC ROC vs epoch waveform



(c) AUC PR vs epoch spectrogram



(d) AUC ROC vs epoch spectrogram

Figure 34: Macro-averaged metrics during training process of Model 4.2

References

[1] J. W. Tukey, "The future of data analysis," Annals of Mathematical Statistics, vol. 33, pp. 1–67, March 1962.

[2] D. R. Raban and A. Gordon, "The evolution of data science and big data research: A bibliometric analysis," Scientometrics, vol. 122, 01 2020.

[3] M. Hilbert and P. Lopez, "The World's Technological Capacity to Store, Communicate, and Compute Information," Science, vol. 332, no. 6025, pp. 60–65, 2011.

[4] J. Foote, "A similarity measure for automatic audio classification," 1997.

[5] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," IEEE Transactions on Speech and Audio Processing, vol. 10, pp. 293 – 302, 08 2002.

[6] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet, "Semantic annotation and retrieval of music and sound effects," IEEE Transactions on Audio, Speech, and Language Processing, vol. 16, pp. 467–476, 2008.

[7] S. R. Ness, A. Theocharis, G. Tzanetakis, and L. G. Martins, "Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs," Proceedings of the 17th ACM international conference on Multimedia, 2009.

[8] M. D. Hoffman, D. M. Blei, and P. R. Cook, "Easy as cba: A simple probabilistic model for tagging music," in ISMIR, 2009.

[9] M. I. Mandel and D. P. Ellis, "Multiple-instance learning for music information retrieval," in ISMIR, 2008.

[10] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green, "Automatic generation of social tags for music recommendation," in NIPS, 2007.

[11] E. Coviello, "Automatic music tagging with time series models," UC San Diego Electronic Theses and Dissertations, 2014.

[12] J. Lee, J. Park, K. L. Kim, and J. Nam, "Sample-level deep convolutional neural networks for music auto-tagging using raw waveforms," 2017.

[13] K. Choi, G. Fazekas, and M. Sandler, "Automatic tagging using deep convolutional neural networks," 2016.

[14] Y. Bayle, M. Robine, and P. Hanna, "Toward faultless content-based playlists generation for instrumentals," 2017.

[15] H. Lee, P. Pham, Y. Largman, and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," pp. 1096–1104, 01 2009.

[16] H. Bahuleyan, "Music genre classification using machine learning techniques," CoRR, vol. abs/1804.01149, 2018.

[17] J. Reed and C. H. Lee, "A study on music genre classification based on universal acoustic models.," pp. 89–94, 01 2006.

[18] A. Tsaptsinos, "Lyrics-based music genre classification using a hierarchical attention network," CoRR, vol. abs/1707.04678, 2017.

[19] A. Sadovsky and X. Chen, "Song genre and artist classification via supervised learning from lyrics," 2006.

[20] A. Kumar, A. Rajpal, and D. Rathore, "Genre classification using word embeddings and deep learning," in 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 2142–2146, 2018.

[21] H. Xie and T. Virtanen, "Zero-shot audio classification based on class label embeddings," 2019.

[22] H. Xie and T. Virtanen, "Zero-shot audio classification via semantic embeddings," 2021.

[23] E. Law, K. West, M. I. Mandel, M. Bay, and J. S. Downie, "Evaluation of algorithms using games: The case of music tagging,"

[24] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset.," pp. 591–596, 01 2011.

[25] D. Bogdanov, M. Won, P. Tovstogan, A. Porter, and X. Serra, "The mtg-jamendo dataset for automatic music tagging," in Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019), (Long Beach, CA, United States), 2019.

[26] M. Won, A. Ferraro, D. Bogdanov, and X. Serra, "Evaluation of cnn-based automatic music tagging models," 2020.

[27] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," Synthesis lectures on artificial intelligence and machine learning, vol. 3, no. 1, pp. 1–130, 2009.

[28] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," IEEE transactions on pattern analysis and machine intelligence, 2020.

[29] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, p. 115–133, 1943.

[30] P. Kidger and T. Lyons, "Universal approximation with deep narrow networks," CoRR, vol. abs/1905.08539, 2019.

[31] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," 2018.

[32] A. K. Bhoi, P. K. Mallick, C. M. Liu, and V. E. E. Balas, "Bio-inspired neurocomputing," Studies in Computational Intelligence, 2021.

[33] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," 03 2019.

[34] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[35] D. R. Wilson and T. R. Martinez, "The need for small learning rates on large problems," vol. 1, pp. 115 – 119 vol.1, 02 2001.

[36] K. P. Murphy, Machine Learning: A Probabilistic Perspective. The MIT Press, 2012.

[37] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," J. Mach. Learn. Res., vol. 12, p. 2121–2159, July 2011.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[39] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," CoRR, vol. abs/1502.03167, 2015.

[41] T. Taulli and M. Oni, Artificial Intelligence Basics: A Non-Technical Introduction. 01 2019.

[42] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological Cybernetics, vol. 36, pp. 193–202, Apr. 1980.

[43] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Computation, vol. 1, no. 4, pp. 541–551, 1989.

[44] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in Proceedings of the 20th International Conference on Artificial Neural Networks: Part III, ICANN'10, (Berlin, Heidelberg), p. 92–101, Springer-Verlag, 2010.

[45] N. H. Phong and B. Ribeiro, "Offline and online deep learning for image recognition," in 2017 4th Experiment@International Conference (exp.at'17), pp. 171–175, 2017.

[46] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising with multi-layer perceptrons, part 1: comparison with existing algorithms and with bounds," 2012.

[47] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means clustering algorithm," Applied Statistics, vol. 28, no. 1, pp. 100–108, 1979.

[48] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley, 1990.

[49] K. Sasirekha and P. Baby, "Agglomerative hierarchical clustering algorithm- a review," 2013.

[50] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," The Computer Journal, vol. 16, pp. 30–34, 01 1973.

[51] D. Defays, "An efficient algorithm for a complete link method," The Computer Journal, vol. 20, no. 4, pp. 364–366, 1977.

[52] J. A. T. Machado, D. Baleanu, and A. C. Luo, Nonlinear and complex dynamics. Applications in physical, biological, and financial systems. Springer, January 2011.

[53] Z. S. Harris, "Distributional structure," Word, vol. 10, no. 2-3, pp. 146–162, 1954.

[54] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in ICLR, 2013.

[55] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13, (Red Hook, NY, USA), p. 3111–3119, Curran Associates Inc., 2013.

[56] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," J. Mach. Learn. Res., vol. 3, p. 1137–1155, Mar. 2003.

[57] X. Rong, "word2vec parameter learning explained," 2016.

[58] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018.

[59] J. Pennington, R. Socher, and C. D. Manning in EMNLP, pp. 1532–1543.

[60] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, vol. 41, no. 6, pp. 391–407, 1990.

[61] A. Carvallo and D. Parra, "Comparing word embeddings for document screening based on active learning," in Computer Science Department, Pontificia Universidad Catolica de Chile, 2019.

[62] A. Joshi, S. Karimi, R. Sparks, C. Paris, and C. R. MacIntyre, "A comparison of word-based and context-based representations for classification problems in health informatics," CoRR, vol. abs/1906.05468, 2019.

[63] G. Berardi, A. Esuli, and D. Marcheggiani, "Word embeddings go to italy: A comparison of models and training datasets.," in IIR, 2015.

[64] V. Ganesh, V. Viswanathan, H. S. Kumar, and E. Sivasankar, "Financial sentiment analysis: A study of feature engineering methodologies," in Soft Computing and Signal Processing, pp. 225–240, Springer, 2021.

[65] J. F. Alm and J. S. Walker, "Time-frequency analysis of musical instruments," Society for Industrial and Applied Mathematics, vol. 44, pp. 457–476, 09 2002.

[66] H. Meng, T. Yan, F. Yuan, and H. Wei, "Speech emotion recognition from 3d log-mel spectrograms with deep learning network," IEEE Access, vol. 7, pp. 125868–125881, 2019.

[67] J. C. Steinberg, "Positions of stimulation in the cochlea by pure tones," The Journal of the Acoustical Society of America, vol. 8, 1937.

[68] H. Fletcher and W. A. Munson, "Relation between loudness and masking," Journal of the Acoustical Society of America, vol. 9, pp. 1–10, 1937.

[69] O. Douglas and O. Shaughnessy, Speech communication: human and machine. Addison-Wesley, 1987.

[70] T. Bertin-Mahieux, D. Eck, F. Maillet, and P. Lamere, "Autotagger: A model for predicting social tags from acoustic features on large music databases," Journal of New Music Research, vol. 37, no. 2, pp. 115–135, 2008.

[71] J. Pons, O. Nieto, M. Prockup, E. Schmidt, A. Ehmann, and X. Serra, "End-to-end learning for music audio tagging at scale," 2018.

[72] J. Read, "Recognising affect in text using pointwise-mutual information," Unpubl. M Sc Diss. Univ. Sussex UK, 2004.

[73] Q. Su, K. Xiang, H. Wang, B. Sun, and S. Yu, "Using pointwise mutual information to identify implicit features in customer reviews," in International Conference on Computer Processing of Oriental Languages, pp. 22–30, Springer, 2006.

[74] B. H. Soleimani and S. Matwin, "Fast pmi-based word embedding with efficient use of unobserved patterns," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 7031–7038, Jul. 2019.

[75] G. Bouma, "Normalized (pointwise) mutual information in collocation extraction," Proceedings of GSCL, vol. 30, pp. 31–40, 2009.

[76] N. Ibtehaz, S. M. Sourav, M. Bayzid, and M. S. Rahman, "Align-gram : Rethinking the skip-gram model for protein sequence analysis," 2020.

[77] F. Charte, A. Rivera, M. J. Jesus, and F. Herrera, "Concurrence among imbalanced labels and its influence on multilabel resampling algorithms," in Hybrid Artificial Intelligence Systems, pp. 110–121, Springer International Publishing, 2014.

[78] K. R. M. Fernando and C. P. Tsokos, "Dynamically weighted balanced loss: Class imbalanced learning and confidence calibration of deep neural networks," IEEE Transactions on Neural Networks and Learning Systems, pp. 1–12, 2021.

[79] K. M. Ibrahim, E. V. Epure, G. Peeters, and G. Richard, "Confidence-based Weighted Loss for Multi-label Classification with Missing Labels," in The 2020 International Conference on Multimedia Retrieval (ICMR '20), (Dublin, Ireland), June 2020.

[80] H. Phan, M. Krawczyk-Becker, T. Gerkmann, and A. Mertins, "Dnn and cnn with weighted and multi-task loss functions for audio event detection," 2017.

[81] S. Panchapagesan, M. Sun, A. Khare, S. Matsoukas, A. Mandal, B. Hoffmeister, and S. Vitaladevuni, "Multi-task learning and weighted cross-entropy for dnn-based keyword spotting.," in Interspeech, vol. 9, pp. 760–764, 2016.