Università degli Studi di Milano

UNIVERSITÀ DEGLI STUDI DI MILANO

Emotion Detection in Song Lyrics

Sentiment Analysis and Opinion Mining

Student
Francesco Lazzara

Degree Course
Data Science and Economics

ID
942830

Contents

1 Introduction **1**

2 Research Question and Methodology **2**

3 Experimental Results **4**

4 Concluding Remarks **8**

5 References **9**

1 Introduction

The aim of this project is to build a classifier able to process texts and detect emotions in song lyrics on the basis of the different language styles used by the each author/singer, in part characterized by the genre of the songs. The steps that would be described throughout the following sections can be mainly divided in two macro-stages: *Model Training & Validation* and *Song Classification & Playlist Creation*. The first step is mainly focused on pre-processing of the corpus of tweets contained in the *Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis* (WASSA-2017) training and development datasets [1]. Moreover, as a step of data-preprocessing, we would also perform some *Feature Vectorization* and *Word Embeddings* of the cleaned texts in order to compare two *Neural Network* architectures and investigate which one leads to the highest accuracy. Though, we would limit the analysis to the validation of the two obtained models with the same evaluation metric on the training portion of data points. On the contrary, the optimization step (tuning) of the corresponding hyperparameters of the two classifiers is beyond the scope of this paper, hence it could be considered for future works with an ad-hoc computing framework, able to handle the mathematical complexity of the task.

Next, in the further stage we would exploit the best classifier to perform a *Multi-Label* classification task and predict the emotions of the sentences of each song contained in the *TidyTuesday* dataset [2], which gets additional information about the artists, lyrics and the genre of the playlists to which the songs belong. Coherently, we would inspect the correlation between the emotions and the music genres to check whether certain feelings are more related to some types of songs or if a relevant number of sentences inside a song may lead to the classification of opposite sentiments overall. In the last part, we would build-up a playlist of one hundred songs based on users mood and music type preferences, which would be directly created on my *Spotify Student* account by establishing a connection with the Spotify API. The first macro-phase is enriched by two components: the *Keyword-based* component and the *Learning-based* component. The first component involves text pre-processing, while the second one is mostly focused on building a learning model to predict emotions from text. Coherently, an *Hybrid-based* approach would consist in a combination of the two previous components and its main advantage is related to fact that it can yield higher accuracy results (Binali, Wu, & Potdar 2010).

Now, even though the tasks to be performed in this study have been set out clearly, some topic-related references have been considered as a guide throughout the choice of some techniques and approaches to be performed. The referenced literature would provide us some base evidences and results that can be exploited to choose suitable methods for our tasks, even though we are perfectly aware that other sources may be considered. In particular, given that the lyrics classification task is more "*experimental*" by design, most of the insights provided by the literature would focus on the first macro-stage, mostly with respect to the text pre-processing and the metrics used to evaluate the models. As a final remark, for a deeper understanding of all the steps made during the realization of the analysis, with all the corresponding outputs and plots, it would be suggested to read the report in combination with the main *Python* code uploaded on *GitHub*[3]. Moreover, in the python code folder all the classes and functions used for the analysis can be found, with the relative explanation of their behavior, plus some additional exploratory analysis and figures, such as the *WordCloud* or the explanation of some predictions given by the best model, etc.

[1] WASSA-2017 Training and Development Datasets

[2] TidyTuesday Dataset

[3] GitHub folder of the project

2 Research Question and Methodology

The goal of our project is to build a predictor trained to identify emotions in texts, in this case some tweets, with the scope of assigning feelings to sentences in song lyrics, which constitute our corpus in the second part of the analysis. In this context, the question we would aim to answer or to provide some insights, for a deeper understanding of the problem through the discussion of our results and assumptions, is whether there is some tendency of certain music genres to be correlated with particular emotions, even more than one and potentially in contrast (ex. "Joy" and "Sadness"). We could use some maths symbology to clearly describe and define our research problem. Given a **Feature Set** of texts under the form of vector of strings: $\mathcal{T} = \{text_1, text_2, ..., text_N\}$ and a finite **Label Set** described by four precisely identified emotions: $E = \{"Anger", "Fear", "Joy", "Sadness"\}$, we could define a *Training Set* of size m as the set of labeled data points/examples: $\mathcal{S}_m = \{(t_1, e_1), ..., (t_m, e_m)\}$ where $t \in \mathcal{T}$ and $e \in E$. Therefore, for this classification task, we would apply a **Learning Algorithm** $\mathcal{A}$ on $\mathcal{S}_m$ in order to learn a classifier $\mathcal{A}(\mathcal{S}_m) = f$ that belongs to a **Predictor Set** $f \in \mathcal{F}$, defined as a mapping function which connects feature vectors to corresponding labels $f : \mathcal{T} \to E$.

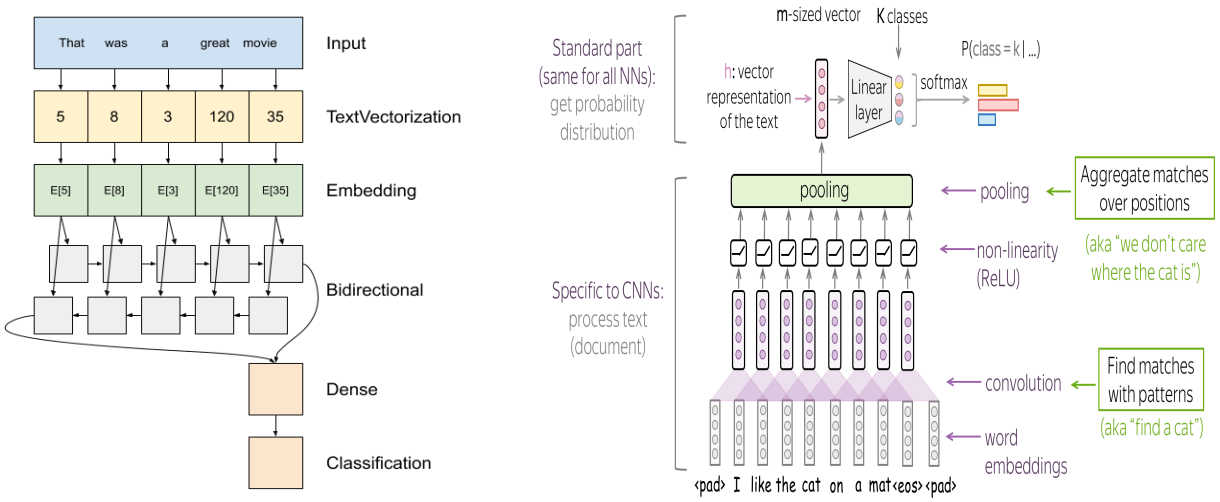
Since we are dealing with vectors of texts that belong to our feature space $\mathcal{T}$, we would need to vectorize their corresponding words into numbers, so that we could obtain a set of integer vectors to be used as input to feed our learning algorithm. This procedure is known as **Word Embeddings** (or *Word Vectorization*) and it is an effective way to convert texts from strings to numbers, before giving them as input for training advanced machine learning models such as artificial neural networks. The strategy that we would use is aimed at encoding documents as dense vectors, which is an efficient approach in contrast with *One-hot encoding* and *Term Frequency-Inverse Document Frequency* (TF-IDF) since they suffer from the issues of dimensionality and sparsity (most indices are zero). Therefore, the first method that we would implement consists in assigning a unique number to each word, so that every sentence can be encoded as a dense vector full of elements, instead of a sparse one, even though it does not capture any relationship between the words given that the distance between two of these kind of vectors would be meaningless. The second technique that we would exploit is the *Embedding layer* provided by *TensorFlow*, accessed through its API *Keras*, included in our neural network architectures, that would return a dense representation of documents such that similar words have similar encoding. Of course, as a general rule the higher is the dimension of the embedding, the larger it takes for the model to learn the data, but at the same time more complex relationships between words can be inferred from the data. Coherently, we would combine these two strategies when training and validating the two neural networks architectures. In particular, we would use a *Tokenizer* to create a dictionary of words as *keys* and index numbers as *values*, which would be used to convert each sentence into a vector of a sequence of token indices (dense) and then be padded so that they all have the same dimension (maximum length of the sequences). Consequently, the embedding layer would generate one vector for each word (token index), so that the input sequence of integers would become a new sequence of vectors with a meaningful notion of distance between them.

In mathematical terms, a **Neural Network** is a combination of multiple matrix-vector multiplications to indicate the information represented in each layer. In particular, by exploiting linear algebra, it is possible to compute the outputs of a fully connected layer with a general expression that works for several instances at ones (Géron 2019):

$$h_{\mathcal{W},\mathbf{b}}(\mathcal{X}) = \phi(\mathcal{X}\mathcal{W} + \mathbf{b}) \tag{1}$$

Coherently with our analysis, $\mathcal{X}$ would represent the matrix of *input features*, namely the sequence vectors of integers, while the *weight matrix* $\mathcal{W}$ would contain all the connection weights except from the bias neuron. In addition, the *bias vector* $\mathbf{b}$ is composed by all the connection weights among the bias and the artificial neurons, so that it would assign one bias term for each artificial neuron, while ϕ is defined as the *activation function*, which is the same for all neurons in a given layer and it is used determine the output of a node, by normalizing it into comparable ranges (ex. unit interval).

For this text classification task we would use two types of models with different architectures: **Recurrent Neural Networks** (RNN) and **Convolutional Neural Networks**. Recurrent neural networks are particularly good at processing sequence of data, such as character strings in a text, since it can store information from previous steps in an hidden layer (known as *Temporal Loop*), which is like a sort of short-term memory for what was in that neuron at the previous steps. However, in the back-propagation through time phase, when the internal weights of the network are adjusted, the gradient tends to get smaller preventing the weights of the early layers to be adjusted so they can learn the data. This is known as *Vanishing Gradients* problem and requires the network to have a long memory, thus the solution is the *Long Short-term Memory Networks* (LSTMs) that can learn long-term dependencies through *gates*, which regulate what information to add/remove to the hidden state. On the other hand, the intuition behind convolutional neural networks applied to texts has some common point with the computer vision domain, mostly about the idea of filtering some words or sentences that could be highly informative and consider them apart from the others, independently with respect to their position. In other words, we aim at detecting some patterns, even though we do not care much about where they would actually be. In practice, a network architecture that can implement this strategy incorporates two layers: a *Convolution*, which finds matches and patterns, and a *Pooling*, which aggregates matches over positions. Accordingly, in the two following figures a summarized version of the two architectures is presented, along with some information about how they work and the output they would produce:



(a) Recurrent Neural Network[4]

(b) Convolutional Neural Network[5]

Figure 1: Neural Networks Architectures used in the *Model Training & Validation* stage

To be consistent with the symbology previously used, since both the neural networks would predict probabilities for each emotion class in E, our classifier would map the feature vectors to a **Prediction Set** $\mathcal{Z}$ different from the label set: $h : \mathcal{T} \rightarrow \mathcal{Z}$. Coherently, during the training phase of the models the sentiment with the highest probability would be chosen as the prediction for a given data point. On the other hand, the classification approach for the second macro-stage, the one regarding the lyrics and the genres of the songs, would be slightly different than the one performed in the first passage. More in detail, we would store the predicted probabilities for each sentence in a *Ragged Tensor* of three dimensions: $Tensor_{[songs,None,emotions]}$. This is because a ragged tensor could store vectors, in each of its dimensions, that do not have a fixed size, which is extremely useful in our context given that the number of sentences defining a song is not the same overall. Therefore, our tensor would store the probability distribution of the emotions for each sentence in a song, which would be used for a multi-label classification task, in combination with an ex-post heuristic rule aimed at determining the label(s) of the songs, based on the frequency of each emotion.

[4]Figure taken from: `https://www.tensorflow.org/text/tutorials/text_classification_rnn?hl=en`

[5]Figure taken from: `https://lena-voita.github.io/nlp_course/text_classification.html`

3 Experimental Results

As described in the first section, two datasets would be used throughout the two subsequent macro-stages. The *WASSA-2017* training set, used in the initial stage, is composed by four columns and 3613 observations. The second column contains the corpus of original tweets posted by some unknown users on Twitter as string datatype, which are labeled according to their intensity, normalized to the unit interval, and one out of four emotions among *anger*, *fear*, *joy* and *sadness*. The tweets were manually annotated by crowdsourcing with some questionnaires that were firstly provided to respondents, then analyzed to obtain reliable scores of the degree of the emotions felt by the speakers (Saif & Bravo-Marquez 2017). On the other hand, the *TidyTuesday* dataset consists of twenty-five features and a total of 18454 data points. For each song, different attributes are given, such as the track and album name, the artist, the lyrics, the playlist genre and some other technical measurement, like "danceability", "loudness" and "tempo". Of course, among the entire set of variables, we would reduce our analysis to a subset that includes the ones suitable for our research that are: the name, the artist and genre of the song along with the lyrics necessary for the multi-label classification task. Moreover, since the obtained classifiers would be trained on tweets written in english, then our research is reduced to the classification of english songs only. Therefore, the size of the dataset is reduced to 15405 possible songs that could be selected to populate the playlists on Spotify, even though the six corresponding music genres appeared to not have an homogenous distribution, since "*Electronic Dance Music*" (560 songs) and "*Latin*" (347 songs) are under-represented with respect to the others.

The analysis starts with the text pre-processing step, in which we would use the `nltk()` library and some *regular expressions* to write a function that would clean up the tweets. In particular, among the other arguments, we would set a string of punctuations (without "#") and two boolean values to remove stop-words and emoji, then the regular expressions would take care about matching mentions to other users ("@user") as well as the digits. Regarding this topic, there is an ongoing debate about whether to remove or not stop-words from texts when performing Twitter sentiment classification, for example by using a pre-compiled list of words offered by some libraries or any other more complex methods. In practice, the key result about this tweet pre-processing technique is that removing singleton words would keep a good trade-off between good performance and low processing time, as well as reducing the sparsity of the feature space (Saif, Fernandez, He & Alani 2014). In addition, as part of pre-processing, we have also written a function that would lemmatize the text and label the words in each tweet with *Part-Of-Speech* (POS) tags, like "noun", "verbs", etc. However, based on the results of Robinson (2016), POS features combined to the text just acted more like noise rather than useful information, so that they did not improve the performance of the classifier with respect to the text features alone. For this reason, we decided to solely focus on the cleaned text when performing feature vectorization to feed the two neural networks, even though it could be interesting to implement other approaches and inspect whether the results are consistent with the literature.

In line with what stated in the previous section, we would tokenize the tweets and covert them in sequence of integers that would be padded to have the same length, corresponding to the maximum number of words in a tweet. With reference to the labels, we would apply *One-hot* encoding to convert the categorical variable into four dummies corresponding to the distinct emotions, which is the standard format for classification tasks with neural networks accepted by TensorFlow. The distribution of the labels in the training set appears to be fairly balanced for three out of four emotions, with the only issue of "fear" being over-represented in almost 32% of observations towards an even distribution of one-fourth each. However, even though we did not find evidence of a substantial imbalance problem, we would decide to stratify the class of labels for the cross-validation task and then compute the *Accuracy* as our evaluation metric:

$$\texttt{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

The accuracy would return an overall measure of how well the model is predicting the correct class for a single observation on average, considering the whole set of data points, even though it does not take into account the class distribution (Grandini, Bagli & Visani 2020). However, by stratifying the labels for the cross-validation algorithm, we would ensure that the distribution of each class in the sample is preserved in all the *k-folds*, so that no severe under-representation of emotions would affect the resulting accuracy computed at each iteration. Next, we would set the standard *Categorical Cross-entropy* as loss function:

$$\texttt{Loss} = -\sum_{i=1}^{4} y_i \cdot \log \hat{y}_i \tag{3}$$

Where y_i is the value of the one-hot encoded label and $\hat{y}_i$ is the predicted probability of the model for the corresponding class. In practice, to compute those two measures, we would write the cross-validation algorithm as a loop that would initially divide the dataset in ten folds, then it would iterate over the split partitions during the training and the validation of the two models for a total of 30 epochs at time. Coherently, setting $k = 10$ is a sensible choice in order to obtain prediction errors that are almost unbiased compared to *Leave-One-Out Cross-Validation* (LOOCV) and bootstrap resampling (Berrar 2018). The resulting performances of the two neural networks architectures, for each chunk of the cross-validation, are summarized in the following table:

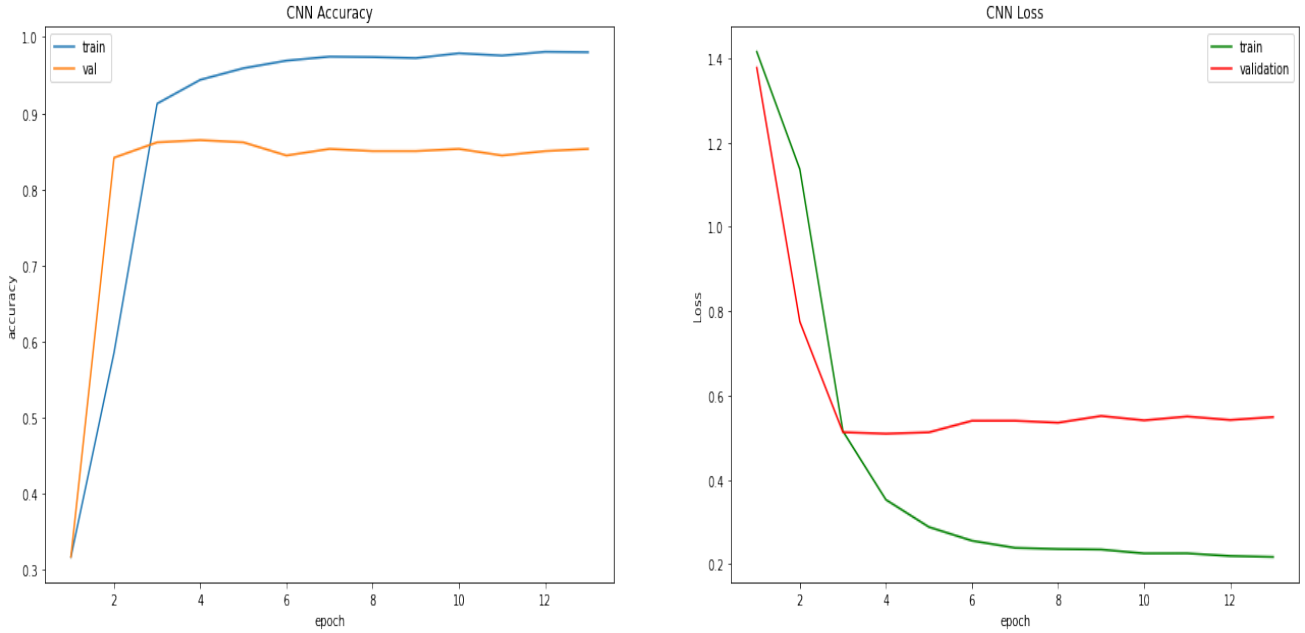
#Fold	RNN - Loss	RNN - Accuracy	CNN - Loss	CNN - Accuracy
Fold 1	1.031	55.801%	0.532	85.635%
Fold 2	0.925	63.536%	0.473	88.122%
Fold 3	1.032	50.553%	0.529	85.912%
Fold 4	0.651	81.718%	0.560	83.657%
Fold 5	1.006	62.327%	0.554	84.488%
Fold 6	1.059	56.233%	0.595	83.934%
Fold 7	0.972	62.881%	0.530	85.596%
Fold 8	0.853	64.820%	0.460	88.643%
Fold 9	0.937	65.374%	0.488	86.981%
Fold 10	1.041	51.524%	0.488	86.427%
Average 10-fold estimate	0.951	61.477%	0.521	85.939%

Table 1: 10-fold Cross-Validation Accuracy and Loss of the two Neural Networks Architectures

As we can see from Table 1, the CNN performed way better than the RNN, with an average level of accuracy that is remarkably high (almost 86%) and resulted to be actually consistent and stable throughout the folds. On the contrary, the RNN suffered from a lot of variance that made the predictions of the classifier pretty unstable and certainly contributed to decrease its overall performance at the end.

At this stage, we would retrieve the CNN architecture and we would re-train the model over the entire training set in combination with an *Early Stopping* rule as a regularization technique. Early stopping is a widely used approach to mitigate overfitting while tracking the performance of the model for every epoch so that, by setting the validation accuracy as monitor, it would stop the training process if no improvement are found with respect to a given threshold. The portion of the examples that would be set as argument of the regularization method is the WASSA-2017 *Development set*, which is available at the same source previously cited, that is composed by 347 tweets. With reference to the regularization rule, we would set a minimum improvement of the validation accuracy

to "`min_delta`$= 0.005$", the maximum number of epochs without gain to "`patience`$= 10$" and we would return the model with the weights from the epoch with the best monitored quantity. Again, the distribution of the classes in the development portion is the same of the training sample, thus we would keep the accuracy as evaluation metric since we are not validating the model anymore. In particular, the highest validation accuracy of the model on the development set was obtained at the fourth epoch, with `val_accuracy`$= 0.865$ that corresponds to a validation loss of `val_loss`$= 0.509$. In the following figure it is displayed the behavior of the classifier in terms of accuracy and loss, corresponding to the the training and development set for each of the 13 epochs.



(a) Validation and Training *Accuracy* of CNN

(b) Validation and Training *Loss* of CNN

Figure 2: Training - Validation Accuracy and Loss of the Convolutional Neural Network

At this point, the final model is obtained and the second macro-phase could start from the recall of the filtered TidyTuesday dataset, with english songs only. As before, we would split the lyrics of each song in sentences and we would clean them, even from wrongly encoded UTF-8 characters and abbreviations, such as "*'til*" or "*'cause*", which are mostly used when writing certain music types, especially rap or pop songs. Next, to predict the emotions of each sentence inside the songs we would need to convert them into padded sequences all with size equal to 24, which was the maximum length of the tweets used to train our neural networks. Therefore, we would limit the classification to sentences composed of at most twenty-four words, instead of cutting the larger ones, mainly for consistency purpose. At this time, we could feed our model with the padded sequences, so that it would return predicted probabilities for each class that would be stored in the ragged tensor. In particular, we decided to store the predictions in a tensor since we prefer to clearly separate the probability distribution of the emotions of every song, hence the second dimension of the tensor would take care of the different number of sentences which can occur among different songs, as explained in the previous section. The output tensor would contain the probability distribution of the emotions for each song, so that for any sentence associated to a song we could retrieve the emotion corresponding to the highest probability and then inspect their absolute frequencies. Now that we have the counts of each emotion for every songs, we need an heuristic rule that would assign them one or more sentiments. A possible approach could be the one to set a cutoff for every song on the proportion of each emotion on the total percentage, such that an emotion would be considered relevant if it is represented by at least a number of sentences equal to the threshold. In practice, since we have four classes we could think about setting the cutoff at `perc_rule`$= 0.25$, so that at least one-forth of labeled sentences are needed to classify a song with a particular emotion. x Now that

the lyrics are classified, we could inspect the combination of genres and the emotions in aggregate, considering the entire dataset. In particular, `(fear, sadness)` appeared to be the most correlated pair of emotions with four music genres: "*Rap*", "*Rhythm and blues*" (R&b), *Rock*" and "*Pop*", with a total of 1103 songs classified with this joint label. Moreover, they resulted to be highly represented even as unique label for the previously cited four genres, with 1284 songs for "fear" and 1199 for sadness, while among the remaining two: 79 "*Latin*" songs were labeled with "sadness" and 130 "*Electronic Dance Music*" with "fear". On the contrary, the second combination of emotions mostly correlated to the four over-represented genres is `(joy, sadness)` that account for a total of 350 songs, even though those two feelings are generally considered as being in contrast to each other. Actually, this fact is consistent with the opposite pair of sentiments `(fear, joy)` that appeared to be the third most frequent combination of two feelings with 371 songs, among all the six genres. Furthermore, since in this the multi-label classification scenario we allowed songs to be potentially assigned to even three or all the four sentiments. Actually this turned out to be a slightly smaller phenomenon and regarded mostly the two combinations `(fear, joy, sadness)` and `(anger, fear, sadness)`, which accounted for 166 and 121 songs, respectively. Again, the latter could be seen as more "coherent" with the general knowledge about the closeness of some feelings, but still we are aware of the event that a song could evoke even conflicting sentiments distributed along its sentences. Among all the combinations, just eight songs contained exactly the same proportion of sentences for each class and, consequently, were classified with all the four emotions, which represent a negligible percentage over the total.

In the last part of the project, we would exploit the Spotify API to build and recommend some playlists considering the user's mood and preferences about music genres, which would be directly added to my student account once the code is executed. The main script is composed by the creation of two classes: one for the authentication to the interface "`SpotifyAPI()`" with my personal credentials (*id* and *secret code*), while the other for the actual creation of the playlists using a temporary *token* and the name of the user. Then, the method "`add_tracks()`" would loop over the dataset according to the feelings and tastes of the user and would match the songs that satisfy the given genre and label constraints. Since the SpotifyAPI allows to retrieve only one-hundred songs at time, then this value would be set as the default size of the final playlist so that 100 songs are drawn at random from the set of the eligible ones. Still, the resulting playlist could contain less than hundred of songs, mostly because the labeled names of the tracks in the dataset could be different from the one in the Spotify archive, especially for the ones registered live at some concerts or remixed. Therefore, the method "`not_found`" would return a list of those eligible songs that should be part of the playlist, but were missed once searched on the music streaming application. Lastly, all the playlists created for different users (friends and family) would be made public and the corresponding links posted in the markdown file "*Spotify Playlists*, inside the *GitHub* folder of the project. To this regard, the following figure is taken from a screenshot of one playlist page from the Spotify desktop app and shows a preview of the songs and album images contained in it, along with a short default description coherent with proposed research:

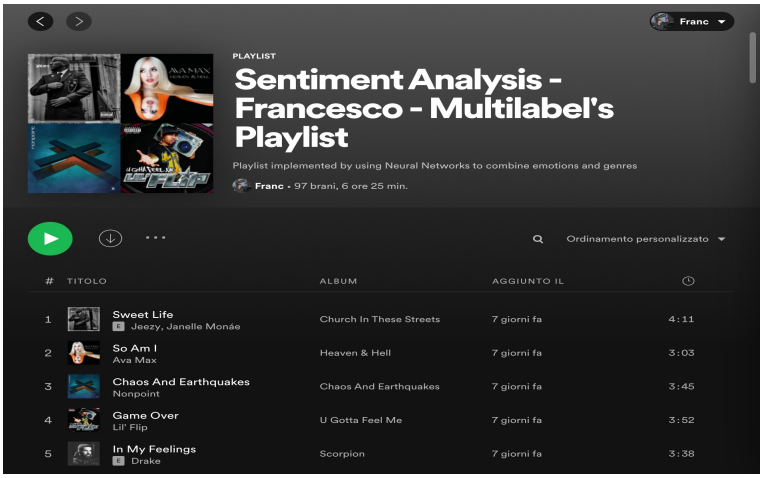


Figure 3: Playlist view from the Spotify desktop app

4 Concluding Remarks

We could say that, in general, when building classifications models over text features there is always some degree of arbitrariness both in the text preprocessing phase and during the experimental stage when tuning the different algorithms to be applied. Here, we provided some insights about the comparison of the performances of two base neural network architectures directly on the field, by evaluating their overall accuracy when assigning emotions to tweets. In this study, we validated both models in order to determine which one would be more suitable for our task, mainly because the emotion detection problem in song lyrics could not be considered as a supervised approach. Hence, given that we do not have at disposal sentiment labels associated to the songs in the dataset, we relied on the "*best model*", hopefully the one picked with the highest accuracy, to classify them and analyze their relationships with the corresponding genres. Of course, the labeled emotions associated to each song would highly depend not only on the classifier generated, but also on the ex-post heuristic rule used in the classification task, thus different approaches may be followed to see whether the obtained results are robust when changing the cutoff value. In practice, the results from the first classification task highlighted that the convolutional neural network (CNN) architecture had a larger accuracy and was not particularly interested by overfitting, rather than the recurrent one (RNN), even though we re-trained it with a regularization technique to let it adapt to more general pattern that could be found in some other unknown set of data points, with the help of a development set.

Now, some considerations could be made about the emotions assigned to the songs and their corresponding genres. At first, "fear" and "sadness" resulted to be the most frequent sentiments associated to the four over-represented genres, thus they tend to be correlated with those two compatible emotions. Moreover, the two sentiments, considered both on an individual and joint level, appeared to be relevant even among the two remaining genres, namely "latin" and "edm". However, this fact did not limited the occurrence of some songs to be classified with multiple sentiments and even opposite ones, which is something highlighted by the pair `(joy, sadness)` being the second most correlated combination to the four main genres. At the same time, this kind of association between differing feelings is found in the triple `(fear, joy, sadness)` being the most represented one for any music genres, apart from "rap", and even more frequent than the `(anger, fear, sadness)` one. Surprisingly, this result could be interpreted as in contrast with what we could have expected by relying on our personal experience and some prior expectations, mainly because of the similarity among these three emotions, which should have placed this triple as the most frequent one among all.

At this point, we could provide some suggestions for future works about this topic. At first, we could experiment a bit about different text cleaning techniques by considering their computational cost over some larger datasets and the performance of different models trained with them. This is because there is no general rule to perform text preprocessing and, since it depends on the topic domain of the texts, then one can provide evidences about the best way to feed text features to algorithms such that they would learn the most from the data, for instance comparing text vectorization against part-of-speech (POS) tagging. Then, provided that there are sufficient computing resources, one could experiment a bit about building complex network architectures and tune them to compare their performances when the hyper-parameters are optimized. As a final note, if some significant relations are found between music genres and the emotions predicted using the lyrics sentences, then we could think about an additional classification problem that consists in predicting the music genres of lyrics contained in an additional set of data points. In other words, we could use the two resulting classifiers, trained one on the tweets while the other on the lyrics labeled with the corresponding genres, to classify the sentiment and the genres of new songs. Hence, we could inspect whether the associations between the track genres and the classes feelings found in this framework are consistent with or differ from the ones predicted for the additional dataset.

5 References

Saif, M. Mohammad & Bravo-Marquez, Felipe. (2017). *In Proceedings of the EMNLP 2017 Workshop on Computational Approaches to Subjectivity, Sentiment, and Social Media (WASSA)*. Copenhagen, Denmark.

Mock, Thomas. (2021).*Tidy Tuesday: A weekly data project aimed at the R ecosystem*.

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras &TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA: O'Reilly Media Inc, 2nd Ed.

Binali, H., Wu, C., & Potdar, V. (2010). *Computational approaches for emotion detection in text*. In 4th IEEE International Conference on Digital Ecosystems and Technologies (pp. 172-177).

Saif, Hassan & Fernandez, Miriam & Alani, Harith. (2014). *On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter*. Proceedings of the 9th International Language Resources and Evaluation Conference (LREC'14). 810-817.

Robinson, Tyler. (2016). *Disaster Tweet Classification using Parts-of-Speech Tags: A Domain Adaptation Approach*. Kansas State University, Department of Computer Science College of Engineering. Manhattan, Kansas.

Grandini, Margherita, Bagli, Enrico & Visani, Giorgio. (2020). *Metrics for Multi-Class Classification: An Overview*. CRIF S.p.A. and Department of Computer Science, University of Bologna.

Berrar, Daniel. (2018). *Cross-Validation*. 10.1016/B978-0-12-809633-8.20349-X.