Predicting Hiphop Popularity: a Spotify case

Samy Zerrouki
HEC Lausanne
MSc. in Finance
samy.zerrouki@unil.ch

***Abstract*—Some people might have heard about the story of Bone music, or how music lovers found a way to defy the authorities and listen to their favorite western music in the 1950's in Russia. Since the arrival of internet, and especially platforms like Spotify, the music industry has boomed. More and more people dream to follow the steps of their idols and become the next Michael Jackson, Nina Simone or Kendrick Lamar. This study focuses on hip-hop music, and aims to find similarities in this genre that could help us predict if it is going to be a hit. The data used were hip-hop songs from 2000 to 2022 provided by spotify's API on which we trained unsupervised learning and deep learning models. Deep learning models seem to perform better than linear models in predicting a song's popularity, and could provide us with the perfect recipe to make a hit. However, popularity is a very subjective term and is subject to other parameters that go beyond the scope of music.**

***Index Terms*—Unsupervised learning, neural networks, classification , spotify**

I. Introduction

Music is an emotional means of communication between humans and is a big aspect of almost every culture on earth. Since it's creation over a decade ago, Spotify has been providing listeners with access to million of songs giving them wider access to music than ever before. Hip-hop isn't one genre, it is a hundred of subgenres, forming one hectic, complicated hole. There is a huge gap between hip-hop's biggest stars and its lesser known actors, yet we still often hear about new, previously unknown artists, making their way to the billboards in the span of a night (eg. Post Malone with White Iverson).

For my project, I wanted to use use machine learning and deep learning to develop a framework that could allow artists to predict the popularity of their song based on its musical attributes. In recent years, members from the community have been more interested in utilizing machine learning to predict song popularity, and it has been given a name - Hit Song Science.

Being a big consumer of hip-hop music myself, I decided to restrain the scope of this project and focus essentially on this genre of music. Focusing only on one genre, and on recent hip-hop songs, we remove the variance in the audio feature data that might jeopardize the correctness of our model. The extracted features relate to acousticness, danceability, energy of our song. These audio features are then used to unsupervised learning and deep learning algorithms. Each algorithm bring interesting results for our quest and on the validity of features to accurately predict the popularity of the song.

Even though popularity is a very subjective term and a lot of factors, external of music, come into account when defining it, I believe we can break down its composition and help newcomers follow their dreams and make their way to the Madison Square Garden.

II. Related Work

While browsing the internet, I found various projects that tried answering this question before me. We will discuss the most relevant researches in Hit Song Science and associated research papers here.

The first research I wanted to mention is Nicholas Indorf's **[1]** brilliant project. He developed a deep learning model to predict the popularity of a song solely based on a thirty seconds audio sample. The best accuracy he obtained was of 59.8 percents and a 0.646 ROC-AUC score which seems really promising.

Another group of Swedish computer scientists **[2]** tried answering the question using machine learning. The audio features they used are similar to the ones I based my study on, but they took into account much more years and genres which made the standardization of the data a big challenge for their research. This issue seems to be common for people answering this question, and is one of the motivations that made me focus on a narrowed data sample.

Last but not least, a group of engineers in the making from Stanford university **[3]** used deep learning to predict hip-hop popularity. Although their project is very well conducted and they produced a working deep learning model, their results were not convincing enough.

III. Data Extraction

A. Challenges

To conduct this project, I was looking for the perfect data set to fit my needs. This data set had to be composed exclusively of hip-hop songs that were released from 2000 to 2022, and had to be long enough to be able to draw from it relevant results for my research. Unfortunately, I couldn't any data set that fitted my needs. I looked through Kaggle data sets, Pitchfork API, and looked at other projects that tried answering a similar question but I couldn't find the perfect data set.

B. Spotify's API

After looking for a data set that didn't exist, I decided to create it myself. Spotify's has a tool called *Spotify for Developers* which allows us to access its API and make requests. There is a library that was created for python which is call *Spotipy* which allows you to get the desired data directly from your python console. I decided to use a code that was developed and made public by Tomi Gelo **[4]** on his repository, which allows you to make requests to Spotify's API and download the data which was later merged in a data frame.

The next issue that I experienced was that Spotify API's search quest limit was modified and went from one hundred thousand to only one thousand results by query. A data frame with one thousand results wasn't enough for me to conduct my research, so I had to find a solution. I tried a *for loop* to make the code run several times but it would always retrieve the same results.

At the end I found a way to make multiple queries at the same time, which allowed me to retrieve data by genre and year at the same time. I ran the code multiple times until I gathered 20394 songs and their attributes.

C. Dataset and features

Spotify feature name	Type	Description
Key	int	Estimated overall key of the track
Mode	int	Modaility (major or minor) of a track
Acousticness	float	Confidence measure of whether the track is acoustic
Danceability	float	How suitable a track is for dancing based on musical elements such as tempo, rhythm stability, beat strength, and overall regularity
Energy	float	Represents a perceptual measure of intensity and activity
Instrumentalness	float	Predicts whether a track contains no vocals
Liveness	float	Detects presence of an audience in the recording to determine if the track was recorded live
Speechiness	float	Detects the presence of spoken words in a track
Loudness	float	Overall loudness of a track in decibels [dB]
Valence	float	Measure from 0.0 to 1.0 quantifying the positiveness (e.g. cheerful, happy, euphoric) of a track
Tempo	float	Overall estimated tempo of a track in beats per minute [BPM]

Description of Spotipy audio features that were used as inputs for the models

Spotipy uses an algorithm to determine the popularity of a song based on the total number of plays and how recent those plays are. Each song's popularity index runs from 0 to 100, with 100 being the most popular. This way of computing the popularity of a song might induce a small bias in cour model. For example, when we looked at the 10 most popular songs of our data set, 9 of the results were from Kendrick Lamar's album that dropped on the 13 of May 2022. Even is the album is a success and Kendrick Lamar's fan have been waiting for 5 years before getting this new release, we can see that these results a biased.

IV. Explanatory Data Analysis

As mentioned in the *Data Extraction* part, the data set features informations about 20394 unique tracks. I described a resume of the audio features in the following figure.

	count	mean	std	min	25%	50%	75%	max
popularity	20394.0	24.755761	16.261749	0.000000	12.0000	23.0000	36.000000	88.000
danceability	20394.0	0.699351	0.141523	0.000000	0.6100	0.7180	0.807000	0.981
energy	20394.0	0.697734	0.150494	0.025500	0.5940	0.7050	0.813000	0.999
key	20394.0	5.335785	3.683785	0.000000	1.0000	6.0000	9.000000	11.000
loudness	20394.0	-6.557615	2.461426	-23.252000	-7.8090	-6.2240	-4.896000	1.089
mode	20394.0	0.564578	0.495824	0.000000	0.0000	1.0000	1.000000	1.000
speechiness	20394.0	0.225046	0.135225	0.000000	0.1040	0.2240	0.320000	0.946
acousticness	20394.0	0.150438	0.175710	0.000002	0.0218	0.0811	0.217000	0.951
instrumentalness	20394.0	0.012977	0.081970	0.000000	0.0000	0.0000	0.000006	0.957
liveness	20394.0	0.224894	0.175993	0.007030	0.1020	0.1560	0.311000	0.996
valence	20394.0	0.540180	0.212499	0.000000	0.3850	0.5500	0.703000	0.980
tempo	20394.0	116.480293	30.697077	0.000000	91.9450	106.7405	139.996750	228.078
duration_ms	20394.0	234420.282436	67414.309676	13140.000000	196180.2500	231188.5000	268067.000000	1240555.000
time_signature	20394.0	3.986418	0.300805	0.000000	4.0000	4.0000	4.000000	5.000
Year	20394.0	2010.976022	6.618705	2000.000000	2005.0000	2011.0000	2017.000000	2022.000

Fig. 1. Descriptive statistics of our audio features

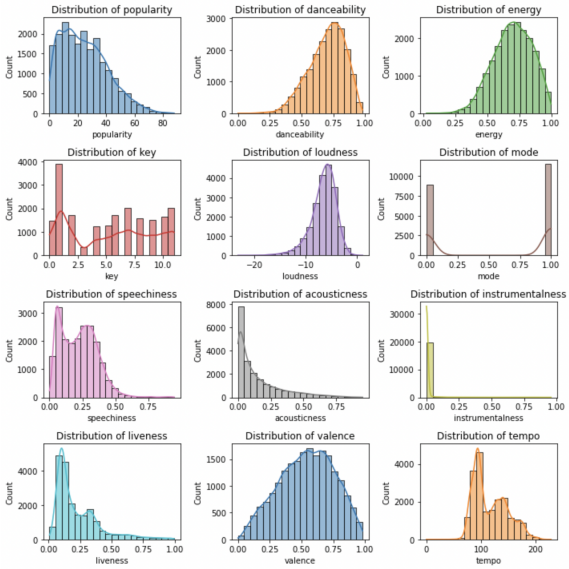

Fig. 2. Distribution of the audio features

From this table we can see interesting information, for example, we understand that 75 percent of our tracks have a popularity of 36 or lower. We can also see that the tracks of our data set have a very low instrumentalness, which can be interpreted as their is a lot of vocals. This is something very current and something we have been expecting when it comes to hip-hop music. This already gives us some insights for our model, as for example, instrumentalness might not be the most relevant feature when we will come down to predicting the popularity.

For the next step, I decided to have a look at the correlation heatmap between all our numerical features to see which ones are the most correlated with the popularity, as it could give us a good start for choosing which features should be included when it will come to creating a model to predict popularity.

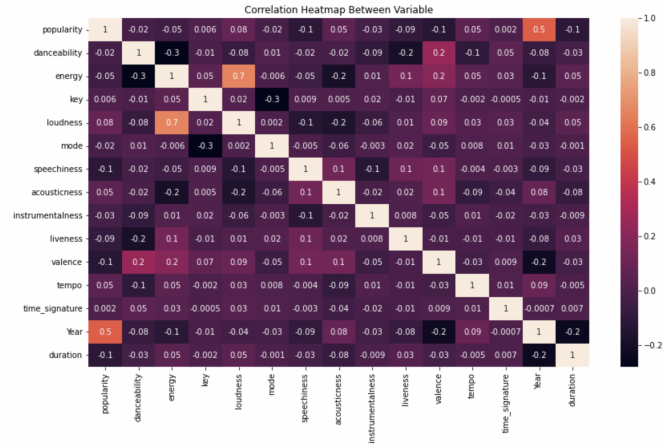

Fig. 3. Correlation Heatmap Between Variables

When having a look at this correlation heatmap, we can identify that the variables that are the most correlated with the popularity are: loudness, speechiness, acousticness and valence. The year and the popularity are extremely correlated as well, but we knew we could expect this result by looking at how Spotipy computes the popularity, and we would like to not take this variable into consideration when constructing our model.

A. *Principal Component Analysis*

The principal components of a collection of points in real coordinate space are a sequence of p unit vectors, where the i-th vector is the direction of a line that best fits the data while being orthogonal to the first i-1 vectors.

The first component is the derived variable formed as a linear combination of the original variables that explains the most variance. The second principal component explains the most variance in what is left when the first component is removed and etc, until the variance is explained. The main focus of this method is therefore a dimensionality reduction that explains the most variance. It can be shown that the principal components are eigenvectors of the covariance matrix.

The first step of a principal components analysis is the standardization of our data. Since the latter is quite sensitive to the variances of our initial variables, if there is large differences between the range of initial variables, the variables with the large ranges will dominate the small ones which will lead to biased results. We perform the standard scaler method on scikit-learn.

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

I wanted to visually represent things in a 3 dimension space, so this is why I chose to have 3 components, and these 3 components explain 46 percent of the total variance. The following figure is the 3-dimensions representation of our data projected according to the 3 principal components.

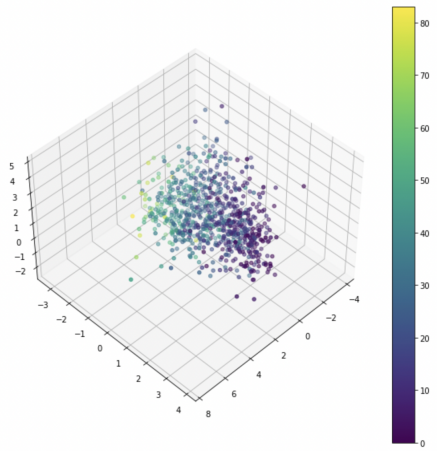

Fig. 4. Representation of our data after a dimensionality reduction with PCA

As we can see from this figure, there is a clear fade in the popularity. The shift is mostly visible on the x and y axis, which are the axis for the first and second componenents of our analysis.

These results are promising, we can interpret from the figure that there are some features that are more important than the others when it comes to explaining the popularity. Having more variability in a particular direction is indicative of something important we want to detect.

B. *K-Means Clustering*

As mentioned in the introduction, hip-hop music is not only one genre, and all hip-hop songs really don't sound like each other. This raises a certain hypothesis, is there a subgenre of hip-hop that is really more popular than the rest. We will try to regroup hip-hop songs together to try and find similarities between them and define subgenres that could help us predict popularity.

K-means is an unsupervised clustering technique that divides unlabeled data into a fixed number of unique categories (the " K"). In other words, k-means identifies observations with similar features and groups them into clusters. A good clustering solution identifies clusters with observations that are more similar within each cluster than the clusters themselves.

The intuition between behind this algorithm is pretty simple, we start by selecting a number of factors K, we then randomly choose an initial center coordinates for each clusters and we then apply a two step process. The first is an assignment step that assigns each observation to its nearest centroid, we then use the update step that updated the centroids as being the center of their respective observations. We repeat these two steps over and over until it converges and we can retrieve our final clustering.

The first challenge of this algorithm is to find unique categories K. We will be using the silhouettes methods. The silhouette coefficient measures how similar data points are within a cluster when compared to other clusters . This coefficient is given by the following equation

$$S(i) = \frac{b(i) - a(i)}{\max[a(i), b(i)]} \tag{2}$$

a(i) is the average distance between i and all the other data points in the cluster to which i belongs. b(i) is the average distance from i to all clusters to which i does not belong.
We will then compute the average silhouette score for each K. The silhouette score is within [-1,1], with 1 denoting the best meaning that a data point is very compact within the cluster where it belongs and far away from the other clusters. Therefore, we should take the number of clusters that maximizes the silhouette score. I plotted the silhouettes scores for k in a range from 2 to 10, and by looking at the plot, we already see that we don't manage to find a number of clusters that is significant enough. Even though I could not find a number of clusters that was significant enough, after standardizing our data using a PCA, I chose K = 6 and tried running the K-means clustering to see if we could still find some similarities in the tracks. The following figure shows the discovered clusters.

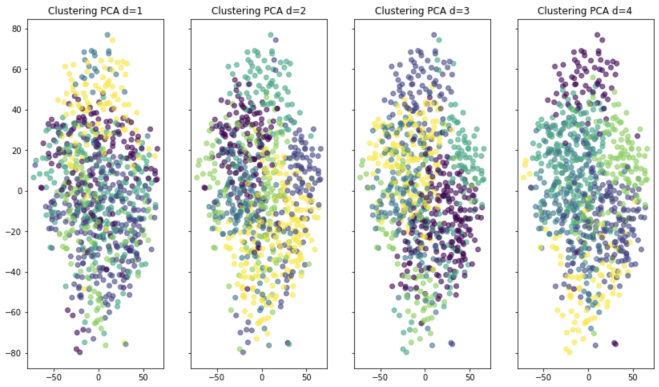

Fig. 5. Discovered clusters according to the dimension of the data

From this graphical representation, we see that we do manage to find some clusters when we reduce the dimension of our data to 4 principal components, so we could interpret that there are subgenres of hip-hop that could be identified in our data. These results are not shocking, we only constructed our data frame with songs from the same genre and that were released in a time period of 20 years, and this is certainly one of the main reasons why we could not find clusters that were significant enough. However, even though these results are somehow encouraging, we will not take them further and explore the hypothesis that a belonging to a subgenre of hip-hop could really help predict the popularity of a song.

V. Linear Models and Supervised Learning

In this section, we will start developing models to see if we can popularity and with which accuracy.

The first technique we will use is the linear model. We define the features that we want to take into account when building this model, and the features that were retained are: *acousticness, danceability, duration, energy, instrumentalness, liveness, speechines, loundness, tempo and valence*.

Linear regression is used for finding linear relationship between the dependent variable (popularity) and the predictors (other features). It looks for statistic relationship between the variables. The goal of this model is to find the line that best fits the data. This line is the one for which total prediction error (distance between the point and the line for all the data points) is as small as possible. The metrics used to evaluate the accuracy of the model is the **R-squared** which ranges from 0 to 1 with 1 indicating that the predictor perfectly accounts for the variation in Y. This metrics is composed of:

1.The regression sum of squares (SSR)

$$\sum_{i=1}^{n}(Predicted\,output - Average\,of\,all\,outputs)^2 \tag{3}$$

This returns information on how far the line of estimated relation is from the horizontal line which is the no-relationship line.

2. The sum squared of errors (SSE)

$$\sum_{i=1}^{n}(Actual\,output - Predicted\,output)^2 \tag{4}$$

3. The total sum of squares (SSTO)

$$\sum_{i=1}^{n}(Actual\,output - Average\,of\,all\,outputs)^2 \tag{5}$$

The R-squared coefficient is given by the following relationship:

$$R^2 = 1 - \frac{SSE}{SSTO} \tag{6}$$

Remember when we plotted the correlation heatmap? We saw that four features seemed more useful to predict the popularity. Therefore, our dependent variable for this model will be the popularity, and the predictors will be ['loudness','speechiness','acousticness','valence']. The reason why we only choose these four features is because if we include all the information we have in our mode, this will result in over-fitting the model.We started our model by splitting our data into two sets, a training set that is useful for model fitting, and a testing set for estimating model's accuracy. This operation was performed using a scikit-learn formula. We chose a 80/20 split for training/testing which refers to the Pareto Principle **[5]**. We then implemented the models and we got a R-squared coefficient of 4.87% which is very low. We then computed other statistics to learn more about the accuracy of this model, and we got the following results, MAE = 12.98, MSE = 253.34 and RMSE = 15.92. To get a better insight on the accuracy of our model , we used cross validation. Cross validation is a technique for assessing how the statistical analysis generalises to an independent

data set. It is useful to evaluate machine learning models by training several models on the subsets of available input data and evaluating them on the complementary set of data. We perform a K-fold cross-validation and set the K = 5 which yielded an accuracy of 8%. We then iterated this method for K in a range of 5 to 10. The results obtained are summarized in the following table:

K	Mean r2	Std r2	Predicted R-Squared
5	0.04415	0.0028	0.0442
6	0.0442	0.00228	0.0442
7	0.04399	0.002066	0.0441
8	0.04388	0.00542	0.0441
9	0.0437	0.00371	0.0441
10	0.0439	0.0035	0.0441

As we can see from this table, our model is not really accurate. This might be due to the features we chose the predict the popularity. As a matter of fact, we made this choice of predictors solely due to correlation between the popularity and the audio features. We now want to know which features are the most important when it comes to predicting the popularity, and we will try to compute the features importance, to see if it could help us achieve a better accuracy, using two methods.

A. Decision Tree Regressor

This approach might be used to address problems like regression and classification. In the shape of a tree structure, a decision tree constructs classification or regression models. It gradually cuts down a dataset into smaller and smaller sections while also developing an associated decision tree. The goal of the decision tree algorithm is to learn basic decision rules from past data and use them to forecast the class or value of a target variable.

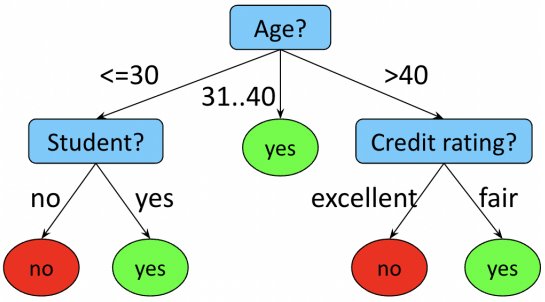

Fig. 6. Example of a Decision Tree

We trained this model with X being all of our features and y being the popularity of the song. To obtain the best depth for this decision tree, I used cross-validation again, and plotted the Root-Mean Squared Error for a depth in range of 1 to 20, I then chose the depth that would give me the lowest RMSE.

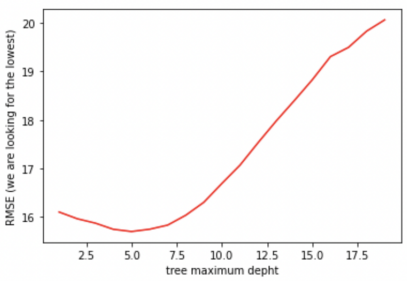

Fig. 7. Plot to look for the best depth

As we can see from this figure, the lowest RMSE is obtained for a decision tree depth of 5, and the RMSE that results from this model is equal to 15.696. Which is not a great improvement from what we had with the linear model.

B. Random Forest Regressor

Bootstrapping is the process of sampling subsets of a dataset at random across a specified number of repetitions and variables. After then, the findings are averaged to get a more strong result. An applied ensemble model is an example of bootstrapping.

The bootstrapping Random Forest algorithm combines ensemble learning methods with the decision tree framework to create multiple randomly drawn decision trees from the data, averaging the results to output a new result that often leads to strong predictions/classifications.

For this Random Forest Regressor, I used the sklearn module for training my random forest regression model. I decided to use 150 estimators and a maximum depth of 4. Then I computed the Out-Of-Bag score of this regressor, which is the number of correctly predicted rows from the Out-Of-Bag sample (rows from our initial data that are not taken for the bootstrap sample) and the RMSE so we could compare it to the one for the linear model and for the Decision Tree Regressor. I got an OOBscore of 0.1934 and a RMSE of 14.6728 which is slightly better than what we found in the other models.

Now, we will have a look at the features importance computed by both of our models.

Decision Tree		Random Forest	
feature	**importance**	**feature**	**importance**
Duration	0.350264	Duration	0.122922
Valence	0.153927	Tempo	0.111128
Speechiness	0.138884	Loudness	0.110337
Liveness	0.087732	Valence	0.109827
Tempo	0.083580	Speechiness	0.104799
Loudness	0.071225	Liveness	0.102140
Acousticness	0.060988	Acousticness	0.101849
Danceability	0.047609	Danceability	0.099778
Energy	0.005792	Energy	0.099343
Instrumentalness	0	Instrumentalness	0.037878

We can compare these results with the features we chose to use when we first computed our linear model. As we can see, the most important feature according to both models is the duration of a song, we can see similarities of the duration of hit songs. As we can see, the features we selected when

doing our linear model are not the same as the top four most important features in both models.

To conclude this part, it was expected by looking at the related work that popularity was not a linear combination of the audio features of a song. Therefore it is normal that we don't manage to find a very accurate model to help us predict it. But do we really need to now what would be the exact popularity of a song, or is knowing how well it would be received enough for our study?

C. Classification

Rather than using regression to predict the exact popularity of a song, it could be useful enough to just predict if the song is going to be popular or not.

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y). We start by cutting our data frame in three parts. The popular songs, the medium songs, and the unpopular ones. A new column is created, in which we will store the popularity score of a song. If we recall the statistical description of our data set, we remember that 75% of the data had a popularity score that was between 0 and 36. Therefore, when doing this operation, we obtain the following repartition: 63.6% of the data is considered unpopular, 33.0% is considered medium, and 3.4% is considered popular. For the moment there is no need to convert the popularity score to an array, but, this will be the case later. We start by performing a train test split. This time it is a bit trickier, we have to create a training and a testing data set while being careful for these sets to be composed of tracks from each category. So we perform a train test split on each of the data sets, then we merge the results into a big testing dataframe and a training one. Then, we need to be careful again with how much features we will use to predict the popularity, because taking too much of them will result in overfitting our model, which we want to avoid. Therefore, because we want to compare the results to the one we found in the other subsections, we will choose the same features to predict the popularity. These features are ['acousticness','loudness','speechiness','valence']. We will then use a new classification algorithm. There is a lot of classification algorithms available now but it is not possible to conclude which one is superior to other. It depends on the application and nature of available data set. The k-Nearest Neighbor method is a lazy learning technique that saves all occurrences in n-dimensional space that match to training data points. When an unknown discrete data is received, it examines the nearest k number of saved instances (nearest neighbors) and returns the most common class as the forecast, whereas real-valued data provides the mean of k nearest neighbors.

The distance-weighted nearest neighbor algorithm uses the following query to weight the contributions of each of the k neighbors based on their distance, providing higher weight to the closest neighbors.

$$\omega \equiv \frac{1}{d(x_q, x_i)^2} \tag{7}$$

We ran this model a first time using a single neighbors to have a taste of how well it would perform, and we have a huge upgrade comparing to the other models. We now find an accuracy of 57.42%. This result can be improved by choosing the right number of neighbors. For this, we create a for loop that will compute the accuracy of the model for k in range 1 to 40. Then we plot the results, and we select the number of neighbors k that maximizes the accuracy of our model. We obtain the following results.

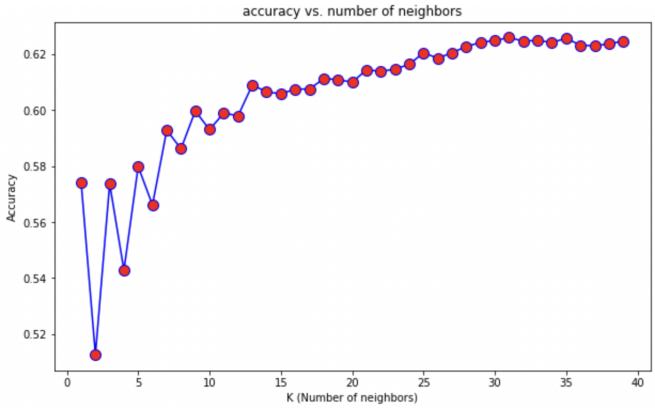

Fig. 8. Accuracy of our model by number of neighbors

As we can see from this figure, we see that the accuracy improves quickly until k = 15, then it changes infinitesimally. The best accuracy of this model is obtained for k = 30, which yields an accuracy of 62.49%.

This final result is a good improve for our research question. But, I was curious to know which kind of popularity score did the model struggle to predict, so I plotted the confusion matrix to have a better view of how he predicts the classes.

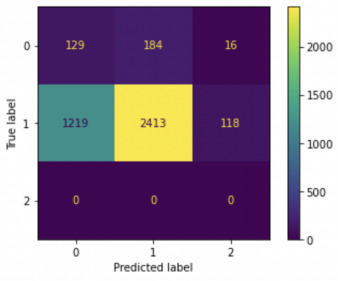

Fig. 9. Confusion Matrix for our KNeighbors predictions

A confusion matrix, also known as an error matrix, is a special table structure that permits visualization of the performance of an algorithm, often a supervised learning one,

in the field of machine learning and specifically the issue of statistical classification (in unsupervised learning it is usually called a matching matrix). The values on the diagonal are the True Positive values. From this plot we can see that our model predicts very well the medium class, but it has issues prediction the not popular and popular ones. This is not a shocking result because we only have 3.4% of the data that is labelled as popular. For the not popular songs though, this seems to be an issue. We want our prediction to tell us if our song is really going to be popular or not and it seems like this model fits most of the data as medium. This result might be due to the use of a KNeighbors classifier, as well as using linear learning algorithms to predict something that we are now sure about, is not a linear combination of its parameters.

VI. Deep learning and Neural Networks

A. Artificial Neural Networks for exact popularity

We determined that the popularity could not be estimated using a linear relationship. If we want our model to be more accurate, we will now have to estimate popularity as a non-linear combination of its predictors. As we mentioned in the Related Work section, most of the work we reviewed decided to use deep learning to predict popularity.

I found a great definition of deep learning by Sarah Narkede **[6]**: Essentially, deep learning is a part of the machine learning family that's based on learning data representations (rather than task-specific algorithms). Deep learning is actually closely related to a class of theories about brain development proposed by cognitive neuroscientists in the early '90s. Just like in the brain (or, more accurately, in the theories and model put together by researchers in the 90s regarding the development of the human neocortex), neural networks use a hierarchy of layered filters in which each layer learns from the previous layer and then passes its output to the next layer. We can say that *Deep learning tries to imitate the activity of neurons in the neocortex's layers*. Deep learning algorithms are useful in many cases, and we already know that big societies like Google, Facebook, or even in our case, Spotify use deep learning algorithms to develop new models which could lead to better filter their consumers needs and propose them with better fitted results. Neural networks are a very common term when it comes to deep learning. There are a lot of them like Long-Short Term Memory (LSTM) that can process entire sequences of data and that are extremely used when trying to make predictions on the prices of crypto-currencies for example. This artificial neural network is very useful when it comes to using time series, and this is why it find a lot of his applications in finance.

In our case, we decided to use a simple artificial neural network provided by the keras API **[7]** in python. We did not meaningful time series data, and the keras API provides us with a nice framework to construct and train our model.

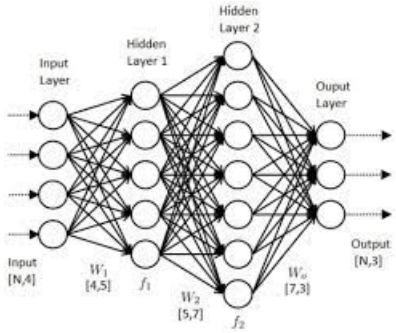

Fig. 10. Example of an Artificial Neural Network with 2 hidden layers

Keras is a Google-developed high-level deep learning API for creating neural networks. It is built in Python and is used to make neural network construction. The first step of our process was again to standardize the data we would be using. Deep learning models learn a mapping from input variables to output variables. We want our inputs to have the same scale, so that we help our model being modelled. If we do not scale the data, this might results in our model learning large weights results, which can lead to the model being unstable and having weak learning performance and sensitivity to input values, resulting in a larger generalization error. Therefore, we use a Standard Scaler from scikit-learn so that the data will look normally distributed (Gaussian with mean 0 and variance 1). Having already splitted our data in a training and testing set, we will now focus on the architecture of our Artificial Neural Network. We trained a neural network with one input layer of 15 neurons that each that 14 values (number of features in our data set) as an input. It is common to have your input layer that has (number of inputs + 1 for bias) neurons. Then we chose the activation method for this first layer, it will be the *Rectified Liner Unit (ReLU)*. If the input is positive, it will output directly; otherwise, it will output zero. Because a model that utilizes it is quicker to train and generally produces higher performance, it has become the default activation function for many types of neural networks.

$$R(z) = \max(0, z) \tag{8}$$

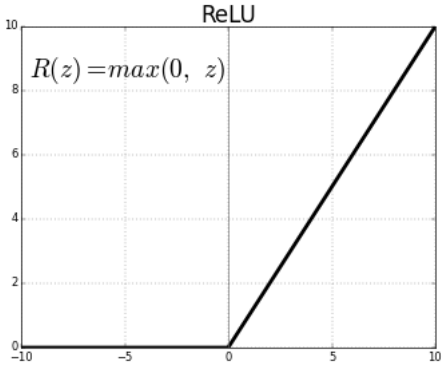

Fig. 11. ReLU activation method

We then move on the our hidden layer. If our data was linearly separable, we would not need any hidden layer, and would not need neural networks in general. For the model, I decided to use one hidden layer. Single-hidden layer neural networks already have a universal representation property: they can fit (nearly) any function by increasing the number of hidden neurons. We trained our model multiple times and a hidden layer with 4 neurons seemed to best best fitted to our model. We also used a ReLU activation function for this layer. And because we only need our output to be the prediction of our popularity, we will only have one output neuron. In total, our model has 294 parameters. When compiling our model, we used an *'Adam'*optimizer which employs a hybrid of two gradient descent techniques: Momentum and Root mean square propagation. It is a popular optimizer in deep learning because it achieves good results fast. The over-fitting issue in our research has always been an issue since the start of this project, and we need to take care of it in this model as well. Therefore, when we compile our optimizer, we set a weight decay of 1e-3 which will be adding a small penalty, usually the L2 norm of the weights, to the loss of the function. This is because our data frame doesn't contain that much observations, if we had million of data points, we would not need this step, but it is not the case so wee need to make sure that the model won't be over-fitting.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_i j \beta_j)^2 + \lambda * \sum_{j=1}^{p} \beta_j^2 \tag{9}$$

We also decide to fit the model, with a validation size of 0.2, on 30 epochs, because after a certain time, our model start over-fitting the data again. To check for the accuracy of our model, we plot the loss and the mean absolute error of the validation and training sets and we get the following results.

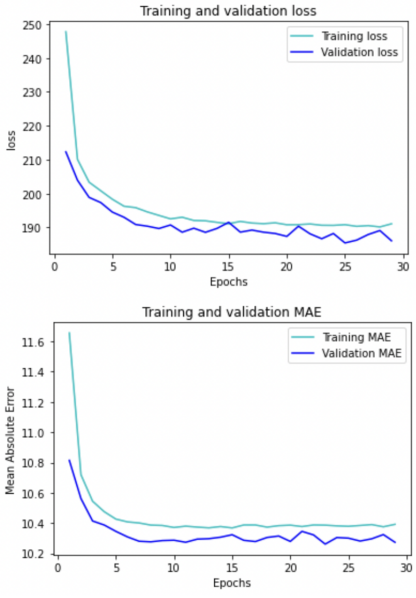

Fig. 12. Neural Network result Validation vs Testing

We also plotted the predictions of our model against the testing set, which gives us the following results.

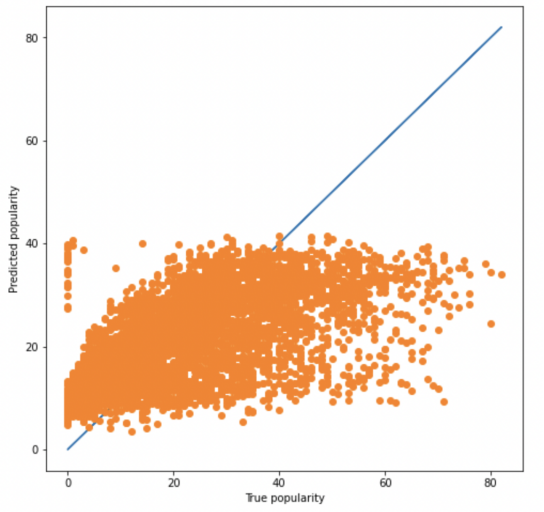

Fig. 13. Neural Network prediction vs test set

As we can see from this plot, our model predicts well the popularity well until the 40-45 threshold, then it tends to underestimate the popularity. This might be due to the fact that our model does not contain a lot of observations with a high popularity score.

When looking at the accuracy of our model, we can see that we made a great improve compared to the linear model, because we have now achieved a coefficient of determination of 28.25%.

B. Artificial Neural Networks and Classification

In the last section, we saw that using a classification algorithm clearly outperforms a regression algorithm in predicting the popularity. We will try to do the same now with a neural network. The first step in this process is encoding our data, we used a label encoder that transform our popularity score into arrays: A popular score will get [0,0,1] as an input, a medium will get [0,1,0] and a not popular will get [1,0,0]. We then modified our output layer so that it will have 3 neurons. We also have to modify the activation method of our output layer to a softmax activation which is an activation function that scales numbers/logits into probabilities. We will then have the probability that the output belongs to a certain class. The softmax activation function is given by the following equation:

$$S(y)_i = \frac{exp(y_i)}{\sum_{j=1}^{n} exp(y_j)} \tag{10}$$

We then plot the loss and the accuracy of our model.

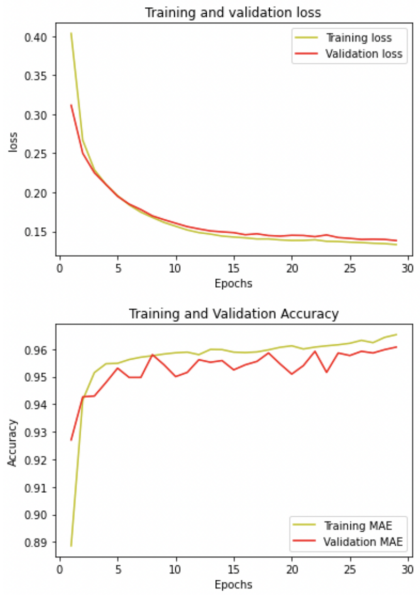

Fig. 14. Neural Network result Validation vs Testing

As we can see, we have now achieved an accuracy between 94% and 96% which is a great result. To get a better insight of of the precision of our model, we plot the confusion matrix. Before doing so, we have to relabel our data because the confusion matrix from sklearn doesn't take arrays as inputs. After computing this, we get the following results.

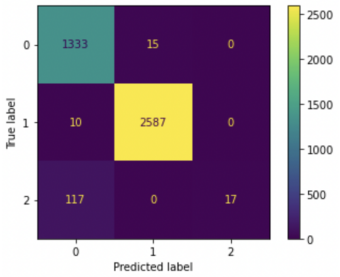

Fig. 15. Confusion Matrix of predictions vs test set

As wee see from this plot, we get way more true positive from the predictions. The only issue still seems to be the popular label.

VII. Conclusion

I feel that a much larger dataset is required to get the desired outcomes. I was unable to construct a suitably enough dataset soon enough due to the result query limit of Spotify's API. One of the main issue of this model was the definition of popularity from Spotipy library. I discover, as in previous research, that while good results are feasible, they are ultimately restricted by the rate of music production, which restricts the size of our current dataset. Popularity is a highly subjective subject to study, and it is not only defined by the audio features of a song. I was recently reading that a Drake's features increases the Spotify streams of an artist by 2783%. Therefore, other factors that may be helpful when trying to predict popularity would be the name of the artists and the lyrics that are used. Hip-hop music depends a lot on the lyrics, in the prior years, a rapper capicity was mainly acknowledged from lyrics he uses in his songs and the story he tells. In future work, it would be very interesting to study these parameters by encoding the artists names and using Natural Language Processing to study the lyrics and get a better insight of the themes discussed in the song construction. Music continues to be a powerful type of media, and as its economic and cultural consequences expand, so will efforts to harness them. This marks the end of my semester project, it has been a delightful experience to work on it and I hope that you enjoyed it as much as I enjoyed making it.

References

[1] N. Indorf "Using Deep Learning to Predict Hip-Hop Popularity on Spotify,"

https://towardsdatascience.com/using-deep-learning-to-predict-hip-hop-popularity-on-spotify-1125dc734ac2.

[2] M. Reiman, P. Örnell, "Predicting Hit Songs with Machine Learning,"

http://kth.diva-portal.org/smash/get/diva2:1214146/FULLTEXT01.pdf.

[3] T. Dairo, H. Kim, N.Wilson , "Predicting Popularity of Rap/Hip-Hop Songs,"

[4] T. Gelo,

https://github.com/tgel0/spotify-data/blob/master/notebooks/SpotifyDataRetrieval.ipynb

[5] Wikipedia, "Pareto Principle,"

[6] S. Narkede, "Understanding Confusion Matrix," https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62

[7] Keras API, https://keras.io/