**AI HARMONY: CLASSIFYING SONGS AS ROCK OR HIP-HOP WITH ARTIFICIAL INTELLIGENCE**

**Abstract**

In this document the binary classification problem is studied with the objective of implementing multiple classification models for a dataset consisting of Hip-Hop and Rock songs, all of it to assess or evaluate each of the models and draw pertinent conclusions.

At the beginning, we focused on preparing the dataset by finding and concatenating multiple datasets which had the target features we were looking for in order to do the classification. After that, a plan to clean and evaluate this dataset is outlined, and its execution is documented and discussed in the Data preparation and results sections.

Some of the main contributions regarding the implementation of this project are the following, a script to create and update the dataset for the classification task; the script to do dimensionality reduction and select the optimal number of components for our dataset; the script to solve the Hyper-parameter optimization problem for our particular task; the script to run and get the metrics/plots for each of the models that we implemented; at last, a simple web application to showcase the classification functionality.

**Introduction**

Problem Description

Classifying songs as 'Hip-Hop' or 'Rock' can be framed as a binary classification problem for the following reasons:

* Binary classification deals with the task of distinguishing between two mutually exclusive categories or classes. In this case, we’re trying to answer the question “Is this song hip-hop or rock?”, defining these two genres as the target classes.
* The models that work with this type of problem require a labeled dataset where each example is associated with one of the two classes, which can be readily collected for hip-hop and rock genres.
* Many common machine learning algorithms, including logistic regression, decision trees, and support vector machines, are readily available and efficient for implementing music genre classification and are well-suited for binary classification tasks.
* Binary classification is conceptually simpler than multi-class classification. It typically requires fewer parameters and can be easier to implement, which can be advantageous when designing and training machine learning models for genre classification.

Questions of interest

* Is this song hip-hop or rock?
* What are the most influential aspects in the classification of a song? Which indicator determines whether an attribute is influential or not?
* How do you address class imbalance, as there might be more songs from one genre than the other in the dataset?
* Are there significant differences between rock and hip-hop liveness? Are there significant differences between rock and hip-hop tempo? Are there significant differences between rock and hip-hop acousticness? Are there significant differences between rock and hip-hop danceability? Are there significant differences between rock and hip-hop energy?

Previous work

The main obstacle when trying to classify music into a genre is the way said music is represented. Firstly, music can be represented in Mel Spectrograms, which visualize audio signals based on their frequency components. Secondly, some audio data can be represented in other specifications such as BPM, loudness, acoustics, “energy,” “liveliness,” etc. This is an easy way to describe each song numerically. Finally, we can use music as time-series data (which makes sense as songs unfold over a time scale) using Mel-frequency cepstral coefficients (MFCCs). Each of these data formats has its benefits and disadvantages based on the application. For instance, [time-series](https://www.projectpro.io/article/time-series-projects/444) data would work best for songs when trained with LSTM or GMM type models.

In (Chillara et al., 2019) we evidence the various authors that proposed their own solutions to the problem, analyzing mainly the audio signals or looking at the music as a time series. However, nowadays the most common solution to the problem is to look at the different properties of the audio. Mainly, looking at the numerical component of the audio signals.

In terms of algorithms, there is a wide selection to be taken in count. Specifically, when using the numerical values of the audio signals and solving the classification problem, we have the following:

* Multiclass Support Vector Machines
* K-Means Clustering
* K-Nearest Neighbors
* Convolutional Neural Networks

**Theory**

Some artificial intelligence models that were considered suitable to solve this particular song classification problem are explained below.

* Random Forest is a machine learning model that uses an ensemble method, since it combines the output of a large number of small decision trees, called estimators. Each decision tree in a Random Forest starts with a simple question and asks more questions based on the answers to the previous ones. These questions are the decision nodes in the tree, which help to split the data. Each individual tree attives to its own predictions, which are later combined to make a single final decision (Wood, n.d.). This model was considered because it’s very flexible and easy to use for both classification and regression problems.
* K-Nearest Neighbors (KNN) is a machine learning model that can be used for predicting categories or values. The ‘K’ in KNN is the number of neighbors that you choose, and ‘nearest’ means that the algorithm looks at the closest instances based on the distance from the test instance to the training instances. This model works by looking at how similar data points are to each other, since it assumes that data points that are similar tend to have similar outcomes. When it needs to make a prediction for a new data point, it looks at the ‘K’ most similar data points from the training data (the ‘neighbors’) and makes a prediction based on them (Nelson, 2020). This model was considered because it doesn’t involve any training period since the data itself is a model that will be the reference for future prediction, and it’s easy to use since it requires only two hyperparameters, and it has high accuracy (Zilliz, 2022).
* Logistic Regression is a machine learning model that is mainly used for binary classification tasks. It combines input values using certain weights to predict a binary value, the probability of an event, outcome or observation happening (Bonthu, 2021). The logistic function, also known as the sigmoid function, takes any real value and turns it into a value between 0 and 1. This function is shaped like an ‘S’ and can never reach the exact values of 0 or 1 (Brownlee, 2020). This model was considered because in a low dimensional dataset with a sufficient number of training examples, it is less prone to overfitting, and because is one of the simplest machine learning algorithms, easy to implement, and provides great training efficiency (Grover, n.d.).
* Decision tree is a machine learning model that works by asking questions about the data. More specifically, it starts with a single question and then splits into branches that can hold potential answers. The branches then lead to decision nodes, which ask more questions that lead to more outcomes that help narrow down the possibilities until a final prediction is made (IBM, n.d.). This model was considered because it’s generally quite intuitive to understand and easy to interpret (Attard, n.d.), and since they show all the possible outcomes of a decision, they are great for making decisions.
* Naive Bayes is a machine learning model that makes the simplifying assumption that the features of the data are independent of each other. Even though this assumption is often not true in real-world data, it allows Naive Bayes to be calculated efficiently and perform well on a variety of problems. This model operates by estimating the likelihood of each class or label for a given data point using the data's features, by calculating the prior probability of each class using the training data and then multiplying it by the probability of the features given that class. The classification is determined by selecting the class with the highest probability (Tornqvist, 2023). This model was considered because it is easy to understand and implement and shows good performance in real-world applications (Ray, 2023).

**Methodology**

The methodology that is going to be used for this project is Cross Industry Standard Process for Data Mining or CRISP-DM. We will now delve into the way the project will be developed taking into account this methodology.

Business Understanding

We are given a dataset of songs compiled by The Echo Nest, a music intelligence company that offers music services to developers and media companies. Our objective is to analyze and classify the songs as either 'Hip-Hop' or 'Rock' with the information provided. To do so, we must clean up the dataset, visualize it, possibly add new entries to the dataset and prepare the dataset to implement a machine learning solution. We must also apply feature reduction in order to optimize the classification algorithms we are going to use.

Data understanding

1. *Data collection*

In terms of data collection we mainly focus on how we can get more data or songs to add to the initial dataset from *The Echo Nest*. In order to do so, we explore multiple options:

* **Search for music based datasets:** For this approach, we look in websites such as kaggle.com, data.world, github.com and many more in search of a dataset containing songs. Said dataset must have attributes that allow us to do the genre classification properly (genre, duration, date of release, audio wave information, numerical values for the audio wave, etc) and those attributes must match the ones we already have in the dataset given to us. We have [this](https://www.kaggle.com/datasets/geomack/spotifyclassification) as an example dataset we could use.
* **Search for music based APIs:** Once again, we search the internet for popular music apps or databases APIs. In particular, we found the [Spotify API](https://developer.spotify.com/documentation/web-api) and the [Echo Nest million song dataset and API](http://millionsongdataset.com/tasteprofile/). With these, we can do different API calls in order to obtain the information we need. For instance, we can query for songs that have “Rock” or “Hip Hop” as their genre and we can set parameters such as the date of release or the artist. This way, we can acquire a great amount of song data without much of a hassle and add it to our existing dataset. It's worth mentioning that in the case of Spotify, instead of directly making calls to the API we can use libraries in Python such as [spotipy](https://opendatascience.com/a-machine-learning-deep-dive-into-my-spotify-data/) which facilitate the data acquisition process even further.

In specific, we focused on the first alternative given that it is less time consuming and more practical for the time constraints that we have. We were able to obtain two datasets that we can integrate into the original one. The first is ANDRII SAMOSHYN’s [Dataset of songs in Spotify](https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify/data) which we found in Kaggle, it contains 42305 songs from a variety of genres. However for this project we only took into consideration the Trap and Hiphop genre given as these are the ones that help us solve our problem. The other dataset is a smaller one, the [Spotify - All Time Top 2000s Mega Dataset](https://www.kaggle.com/datasets/iamsumat/spotify-top-2000s-mega-dataset) which contains around 2000 songs and we mainly used it to obtain some extra rock songs that we didn’t already have.

1. *Data description*

After merging the dataset from *The Echo Nest* with the *fma-rock-vs-hiphop* dataset, and concatenating the other 2 new datasets we obtained, we created a new dataset with 10878

entries and 15 attributes, with 14 of them being numeric and the other being text.

The process followed in order to come up with this dataset was the following: This dataset is the result of getting rid of the unnecessary variables:

'type', 'id', 'uri', 'track\_href', 'analysis\_url', 'title','time\_signature','Unnamed: 0'

'Index', 'Artist', 'Popularity', 'Year'

'track\_id', 'bit\_rate', 'comments', 'composer', 'interest', 'date\_created', 'date\_recorded','favorites','genres', 'genres\_all', 'information', 'language\_code', 'license', 'lyricist', 'number', 'publisher', 'tags'

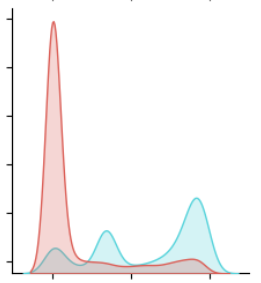
All these variables don’t really give useful information when trying to classify or identify the genre of a song. The remaining variables were adjusted in order to fit them in the model. Firstly, the genres were transformed into binary values (0 for Hip Hop and 1 for Rock) in order to include them in the numeric analysis. Secondly, we looked for duplicate entries in the dataset and removed them. Finally, we used a simple imputer in order to get rid of null values that are present in the different variables. In particular, we replace the nulls with the mean of the variable where said null is present. The two main variables in which this was used were instrumentalness and loudness.

The final dataset variables are as follows: genre\_top, duration, loudness, tempo, speechiness, acousticness, danceability, valence, energy, instrumentalness, liveness.

*Data exploration*

For the data exploration, we take these variables and do an analysis of their distribution and possible correlation between each other:

* Firstly, we obtain the [standard statistical measurements](https://drive.google.com/file/d/1wGpMbpizkNL6TEbxnRUO9CUZBghZD1_Y/view?usp=sharing) for the numerical variables present.
* Secondly, we find out with this new dataset, the distribution of the target variable. In particular we wanna know how many Hip-Hop and Rock songs there are respectively:
  + Hip-Hop: 6129
  + Rock: 4749
* Thirdly, we try to find the relation between each of the variables and the target variable (the genre), for this purpose, we make a [correlation heatmap](https://drive.google.com/file/d/19YMceYz6y6h__2wkfzSLXJd_OD_SM2lJ/view?usp=sharing). Which allows us to see that every variable has a correlation with the genre (except for the “listens” variable).
* Finally, we made a pairplot to compare every pair of variables with each other, taking into account the genre for this comparison. A nice feature of this pairplot is that we can see how the genre is distributed along a single variable. For example, in instrumentalness we can see:



Where the red is Hip Hop and the light blue is Rock. For instance, here we can see that Hip Hop songs have less instrumentals than rock songs. The full pairplot can be found [here](https://drive.google.com/file/d/18wETWTPEWYopSjTMBAKI3URANtc39UHD/view?usp=sharing).

Additionally, we set the goal to make a comparison with the results from this original dataset and an entirely different dataset created from the original by applying Principal Component Analysis (PCA). This allows us to reduce the number of features. It uses the absolute variance of a feature to rotate the data, however, a feature with a broader range of values will overpower and bias the algorithm relative to the other features. To avoid this, we must first normalize our features. There are a few methods to do this, but a common way is through standardization, such that all features have a mean = 0 and standard deviation = 1 (the resultant is a z-score).

A figure that shows the explained variance ratio for each component can be found [here](https://docs.google.com/document/d/1172uamp77SGX2bp4V9T3MaVqBLfs_0Y1Bi-g0cQ0jkg/edit?usp=sharing). In this figure, we obtained 10 components (the same as the number of features in our analysis) and we proceed to plot the cumulative explained variance in order to determine which of these components are actually worth keeping for our modeling. The resulting figure can be found [here](https://docs.google.com/document/d/1172uamp77SGX2bp4V9T3MaVqBLfs_0Y1Bi-g0cQ0jkg/edit?usp=sharing).

Now, we can set a variance threshold, in other words, we can determine how much variance we are willing to accept so that our model can still produce valid results. In this particular case, we went with 85% of the variance. With this threshold, we consider only the first 5 components in our analysis obtaining 87% of the variance in this manner. Following, are the results of the different models while applied to the original dataset and the one with PCA.

Deployment

The was locally deployed as a website, in which the user can enter data about a song and, by using an AI classification model, the website will show to the user the probability of that song to be a hip hop or a rock song.

The selected model for the demo application was Random Forest, and its implementation can be seen in this [video](https://drive.google.com/open?id=1vRsAQ_6yVQ8LbiUEvXhaxujdn9orwJpM).

**Results**

Modeling

For the results, we focus on the results obtained in the different classification models we implemented. To reiterate, we are looking to determine the genre of a song based on the variables that we ended up with in the last section of this document. To do this, we propose a variety of models that are going to be trained and tested with our dataset. The distribution we considered to implement these models is 70% of the data is going to be part of the training dataset and the remaining 30% is the test dataset.

Additionally, to further optimize these models we use GridSearchCV which is a function that allows us to search for the best possible hyperparameters for each one of the algorithms. For example, it can tell us what is the depth in the random forest model that yields the best results (in terms of accuracy of the prediction). On top of that, we use stratified K fold cross validation (with 5 folds) for each one of the models to get a more thorough execution and achieve more precise results.

Following, are the results obtained in each of the models:

|  | **Optimal hyperparameters** | | | |
| --- | --- | --- | --- | --- |
| **Model** | **Without PCA** | | **With PCA** | |
| **Random Forest** | bootstrap: False  criterion: ‘gini’  max\_depth: 4  max\_features: ‘auto’  min\_samples\_leaf: 2  min\_samples\_split: 2  n\_estimators: 25 | | bootstrap: False  criterion: 'gini’  max\_depth: 4  max\_features: 'sqrt'  min\_samples\_leaf: 2 min\_samples\_split: 5  n\_estimators: 48 | |
| **Naive Bayes** | var\_smoothing: 0.572 | | var\_smoothing: 0.00215 | |
| **Logistic Regression** | c: 0.0127427498570  max\_iter: 1000  penalty: ‘l2’ | | c: 0.004832930238571752  max\_iter: 1000  penalty: ‘l2’ | |
| **K-Nearest Neighbors** | algorithm: 'auto'  metric: 'manhattan'  n\_neighbors: 5  p: 2 | | algorithm: 'auto'  metric: 'manhattan'  n\_neighbors: 11  p: 2 | |
| **Decision Tree** | criterion: 'entropy'  max\_depth: 10  splitter: 'best' | | criterion: ‘gini’  max\_depth: 5  splitter: 'best' | |

A comparison between the metrics obtained for each model with the training and the testing set, and with and without PCA with 5 components can be found [here](https://docs.google.com/spreadsheets/d/1RnV6ltN4RQq8LfWZyi4poEmp7hu-zpYK/edit?usp=sharing&ouid=113882557560976941263&rtpof=true&sd=true).

**Results analysis**

Evaluation

To evaluate, we first implement a visualization for the data presented in the tables above. To do this, we make use of the Receiver Operating Characteristic curve (ROC) and its Area Under the Curve (AUC). ROC is a graph that shows the performance of binary classification models. It is used to plot two parameters which are the true positive rate and the false positive rate:

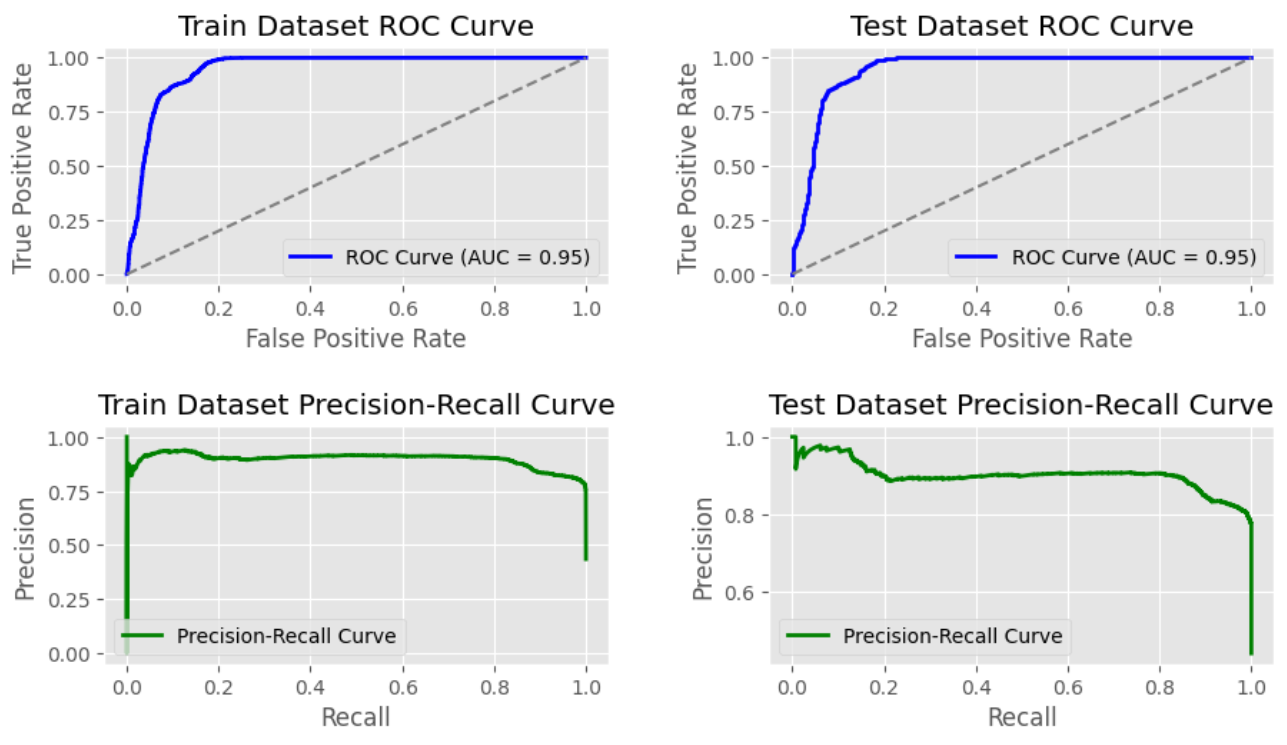
* True Positive Rate = Total Positive / Total Positive + False Negative
* False Positive Rate = False Positive / False Positive + True Negative

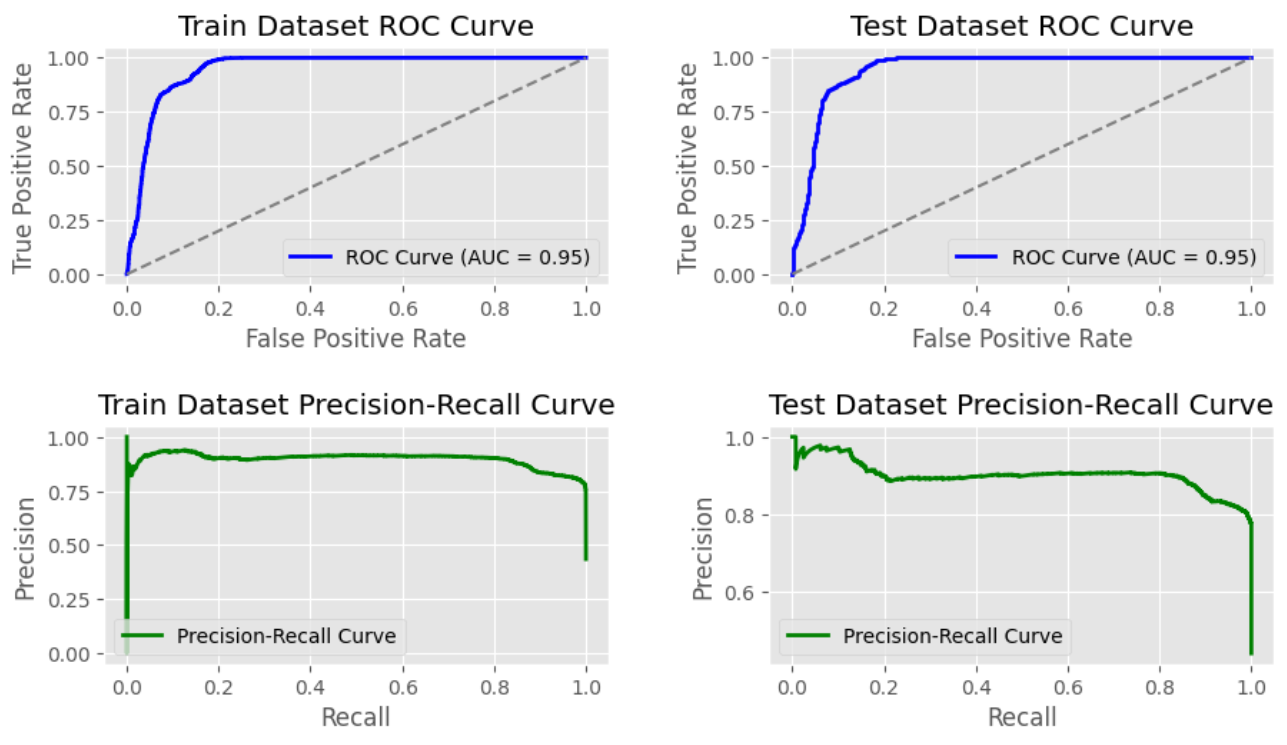
The ROC curve is very similar to the precision and recall curve, but the difference is that instead of plotting the precision and the recall for a given model, it plots the rate of true positives versus the rate of false positives. The AUC is between 0 and 1. A classification model with 100% bad predictions will have an AUC score of 0.0, while a classification model with 100% true predictions will represent the AUC score of 1.0. Alongside this, we use the Precision-Recall curve to determine from which recall we have a degradation of precision and vice versa. Ideally, we want a curve that is as close as possible to the upper right corner (high precision and high recall).

These curves can be found [here](https://docs.google.com/document/u/1/d/1e0i6qdprdn-tp5_ekM_AQWFXw_f6bdXPSYrrT-ZKe8M/edit).

For the sake of the analysis, we are going to use the logistic regression curves to explain so that after we can generalize to every single model.

* **Without PCA:**

****

* **With PCA:**

Firstly, when comparing the results of the original dataset and the modified one we can see a stark resemblance between the two of them. PCA conserves the general shape and form of the curve even through the dimensionality reduction which gives us confidence in the fact that the 87% explained variance is indeed a good representation of the dataset as a whole. Secondly, we can look to the ROC curve and its area under the curve, our goal is to have the curve be as close as possible to the top left of the graph and for the AUC to be as close to 1 as possible, this is because the ROC curve represents the rate of true positives (predictions made correctly) versus the rate of false positives (predictions that were supposed to be false but were classified as true) so, ideally we want to have 0 false positives. In this instance however, we landed with a 0.95 AUC or 95% of the data that was positive was classified correctly.

This value was consistent for both datasets and, in a general sense, the results for the ROC curve were consistent for the majority of the models, given the metrics and results for each one. The lowest AUC was 94% in the Naive Bayes algorithm. However, we must mention that in our dataset, both classification classes (Hip-Hop and Rock) were imbalanced, Hip-Hop had 27% more entries in the data than Rock making it possible that there is a bias in the curve and it’s not giving accurate results.

For this reason, we turn to the Precision-Recall curve which may yield better results given how our dataset is imbalanced. Similar to ROC, this curve gives us an idea of how well our model is performing, specifically, a high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. It can be interpreted as how many of the false (0) predictions were made correctly. The area under the curve gives the average precision value which in the case of logistic regression is 92%. Once again, for the majority of our models this reigned true as the lowest value was 87% for Naive Bayes once again. We can be confident in this curve because this one isn’t affected by the imbalance in the data in the same way that the ROC curve is.

One last thing to highlight from the results is the execution times when applying the models to the original dataset and the dataset obtained from PCA. For instance, when running the random forest model with the original dataset the execution time is 24 minutes whereas with PCA we get a time of 16 minutes, around 40% less time. And we can continue to see this pattern of behavior for the other models as well. Overall, the time of execution when using PCA instead of the original dataset is around 30% on average.

**Conclusions and Future Work**

The main conclusion we draw from this project is that the binary classification problem doesn’t have a one simple way to be approached. It is not a solved problem because there are many techniques and algorithms that one can use to solve it, from to collecting and preparing the dataset properly, to doing dimensionality reduction, to choosing one of the many models and their variations taking into account certain criteria, to applying optimizations like cross validation. It can be very overwhelming, but that is what makes it captivating. Different people may arrive at different solutions because of this and that is fine, it is all in the search of a solution that is capable of solving the problem specification.

During this process, we had our share of mistakes that could have been avoided or corrected if we had more time. For instance, it would have been insightful to plot decision boundaries for multiple pairs of features to see how much each feature contributed to the overall prediction. Also, it would have been interesting to get a more balanced dataset by either adding more Rock songs or by removing some of the hip-hop ones, this way we may have had more accurate results overall. These are some of the possible points that could be tackled in a future iteration of this project.

**Bibliography**

Chillara, S., S, K. A., Neginhal, S. A., Haldia, S., & S, V. K. (2019). Music Genre Classification using Machine Learning Algorithms: A comparison. *International Research Journal of Engineering and Technology (IRJET)*, *06*(05), 851–858.

Wood, T. (n.d.). *Random forests*. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/random-forest

Nelson, D. (2020, August 23). *What is a KNN (K-nearest neighbors)?*. Unite.AI. https://www.unite.ai/what-is-k-nearest-neighbors/

Zilliz. (2022, October 17). *What is K-Nearest Neighbors (KNN) algorithm in machine learning? an essential guide*. Zilliz Vector database blog. https://zilliz.com/blog/k-nearest-neighbor-algorithm-for-machine-learning

Bonthu, H. (2021, July 11). *An introduction to logistic regression*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/07/an-introduction-to-logistic-regression/

Brownlee, J. (2020, August 15). *Logistic regression for machine learning*. Machine Learning Mastery. https://machinelearningmastery.com/logistic-regression-for-machine-learning/

Grover, K. (n.d.). *Advantages and disadvantages of logistic regression*. OpenGenus IQ: Computing Expertise & Legacy. https://iq.opengenus.org/advantages-and-disadvantages-of-logistic-regression/

IBM. (n.d.). *What is a Decision Tree?* https://www.ibm.com/topics/decision-trees

Attard, M. (n.d.). *8 key advantages and disadvantages of decision trees*. Inside Learning Machines. https://insidelearningmachines.com/advantages\_and\_disadvantages\_of\_decision\_trees/

Tornqvist, M. (2023, March 27). *Building a machine learning classifier with naive bayes theorem: A Java example*. Medium. https://medium.com/@mattias.trnqvist/building-a-machine-learning-classifier-with-naive-bayes-theorem-a-java-example-aad2d9f2b616

Ray, S. (2023, July 10). *Naive Bayes classifier explained: Applications and practice problems of naive Bayes classifier*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

*Solved Music genre classification project using Deep Learning*. ProjectPro. (n.d.). https://www.projectpro.io/article/music-genre-classification-project-python-code/566#mcetoc\_1fsm039889

*Music genre classification using CNN*. Clairvoyant. (n.d.). https://www.clairvoyant.ai/blog/music-genre-classification-using-cnn