CS361: Spotify Song Mood Analysis using Classification and Clustering

Achintya Gupta Ankita Kanoji Arnav Sharan Kannan Rustagi

Abstract

We have devised an innovative approach to predict a track's mood based on audio features. We have mainly classified a song into one of four moods – **calm, happy, energetic, and sad**. Our dataset contained some of the important features that helped in classifying songs according to different moods as well as in clustering similar songs together. For **EDA**, we have constructed Heatmaps correlating the features, Radar plots, and KDE plots for better visualization of our data. We attempted to do feature selection using **Principal Component Analysis**. We have implemented **Classification** as well as **Clustering** methods to model our data. The Classification models include **Gaussian Naive Bayes classifier**, **Decision Tree**, **Random Forest Classifier** and **K-Nearest Neighbours**. As for clustering, we have employed the **K-Means** algorithm. After this, we compared the results obtained from both paradigms and attempted to draw conclusions on our dataset.

1. Introduction

1.1. Motivation

"Can we teach a computer to learn how music will make people feel?" We currently use many different methods to classify the music that we listen to, such as genre, artist, etc. For us, however, a song usually induces a feeling in us that cannot be easily explained, but often we can associate certain moods with them. Identifying the mood of a song can improve the existing song recommendation algorithms, which only use the genre and similar artists.

1.2. Target Problem

We have a large collection of digital music, and while existing methods to categorize the songs are quite complex, taking into account various metrics, they often fall short in capturing the human emotions invoked by the music. Quantifying such data through statistical methods can prove to be quite a challenging task, and tackling this obstacle through Machine Learning methods is a worthwhile problem to solve.

1.3. Objective

The primary objective of this model is to accurately identify and categorize the mood associated with a user given song. Spotify is the largest music streaming platform in today's world. We focus on improving its capability of recommending a song based on the user's mood. If a user is listening to a song, our model will predict the mood associated with that song and spotify can recommend songs that come under the same mood category. This will highly improve user's engageability and will lead to increased satisfaction and loyalty. Only genre based and artist based suggestions may not resonate with the user's mood, thus integrating mood based suggestions will enhance precision and personalization in song recommendations.

2. Methods

2.1. Data Scraping

For data scraping, we followed the given process-

- We created an account on the platform 'spotify for developers', followed by the creation of a dummy app, whose client credentials we will use to extract the required information.
- Further, we have used the spotipy library of python to help us extract the song features. This library ultimately helps us by making a call to the spotify web API which in turns returns the features of each song in the playlist given to it as input.
- One major challenge we encountered here is that there is a certain limit set at the number of API calls that will occur in a time frame of 30s by the dummy app we have created. Hence, we had to extract the information in batches of 100 songs, this was a tedious process.
- Then, we ultimately combined the data into a csv file.

2.2. Data Collection

We had initially planned to make our own dataset using the spotipy python library which fetches the data from several playlists labelled as 'Calm', 'Happy', 'Sad' and 'Energetic using the Data Scraping process above. We were faced with the following issues-

- The standard playlists made by Spotify which seemed to be accurate were fairly small (usually 50-100 songs) which did not provide for proper accurate training of the model.
- Large user made playlists had a lot of variance, where songs overlapped as well as could be incorrectly labelled based on the biased preferences of the user who created the playlist.
- We had to remove the duplicates, since our model does not handle multi-label classification. Furthermore, on plotting the data, we realised that the given playlists had roughly the same mean values for all the features. This might again be due to variety in Hindi and English songs and differences in how someone personally classifies different songs.
- This creates a need for manual labelling for data, which is not feasible for large datasets for the given timeline of the project.

Hence we went forward with a dataset found online.

2.3. Exploratory Data Analysis

Firstly, we dropped those columns from the dataset which were not relevant for predicting the mood.

- We observed that the number of calm songs were almost double the number of songs of any other mood label. Hence, we chose to randomly drop a certain number of calm songs from the dataset to make the distribution of songs of each label equal.
- We used a heatmap plot to be able to visualize the correlation between all the available features. We found that there is not much correlation in our feature set.

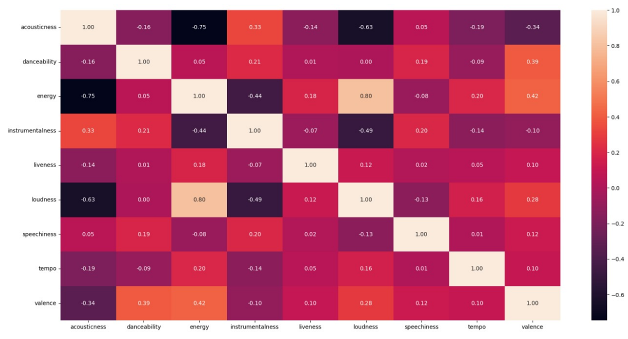

Figure 1. Feature Correlation Heatmap

- Next, we made plots for how all the songs are distributed for each feature, i.e., the frequency of songs for different feature values.
- Building upon the previous step, we also made Kernel Density Plots(KDE plots), wherein we can visualise the data using a continuous probability curve, plotted separately for each feature. Then we also plotted class conditional distribution about the features for each class and every feature, wherein we could see how for most features it resembled the Gaussian distribution, this is why we decided to add Naive Bayes Classifier as a modelling method as well and see its performance.

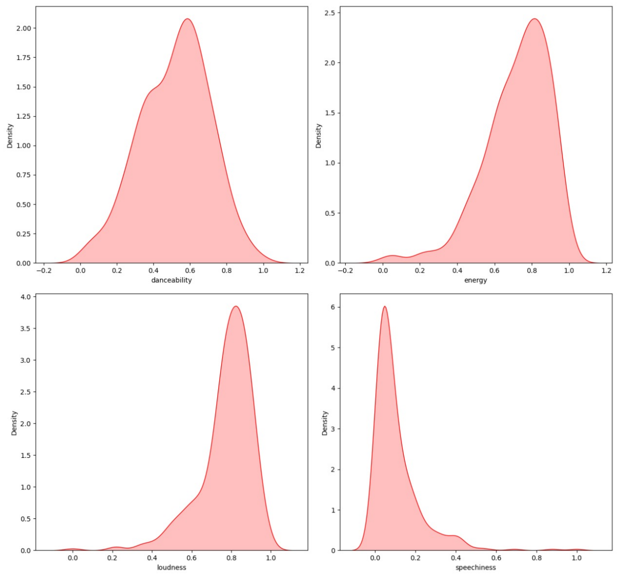

Figure 2. Class Conditional Density Plots (for Happy)

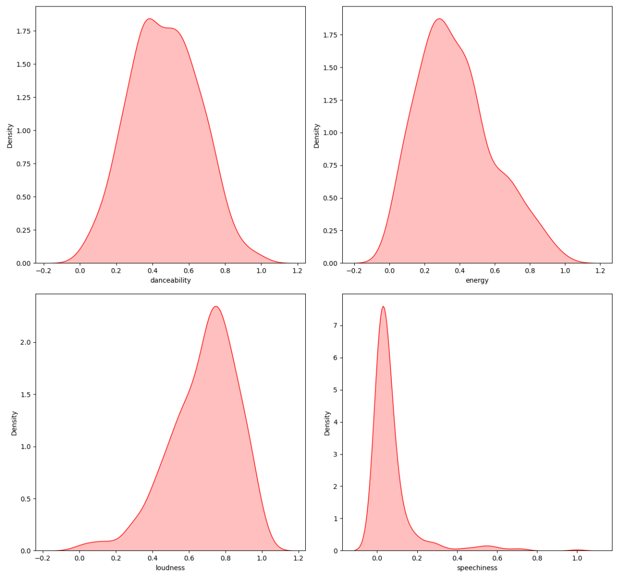

Figure 3. Class Conditional Density Plots (for Sad)

- For each feature, we have shown its average value for every class to see the differences and similarities between our defined classes in terms of the given features.
- We segregated the data based on classes. Then for each class, we first normalised each feature using min-max normalisation. Then, we took the mean and median values of all features and plotted a radar chart to visualize what average values all features take for each class and how they are different for all the classes. The radar charts based on mean and median were fairly the same for every class.

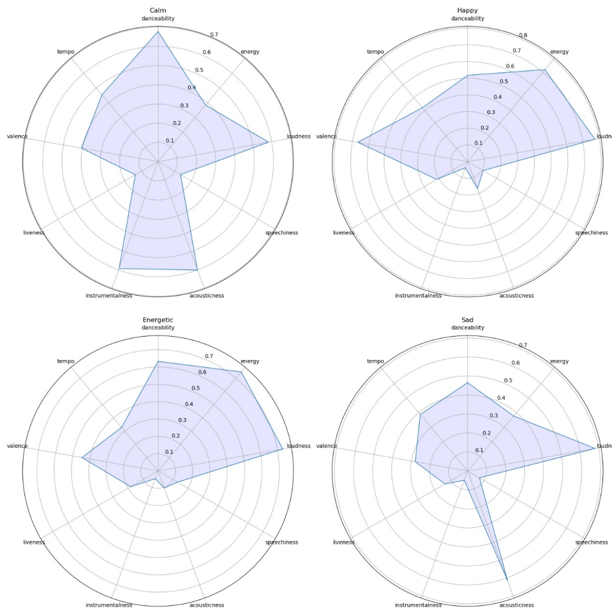

Figure 4. Radar Plots for each mood

2.4. Feature Selection using PCA

It is important to gauge whether the features we have selected are useful. We checked if there was a possibility of further reduction of features and finding a set of features that could accurately classify the given data without significant loss of information. We utilized Principal Components Analysis (PCA) to explore feature selection opportunities within the dataset. PCA, a widely used technique in machine learning for dimensionality reduction, efficiently transforms original features into uncorrelated variables known as principal components.

2.4.1. Implementation and Observations:

We calculated the eigenvalues corresponding to the dataset using the numpy's inbuilt eig to find the effectiveness of PCA. Once we had the eigenvalues and corresponding eigenvectors, we could calculate the best principal components

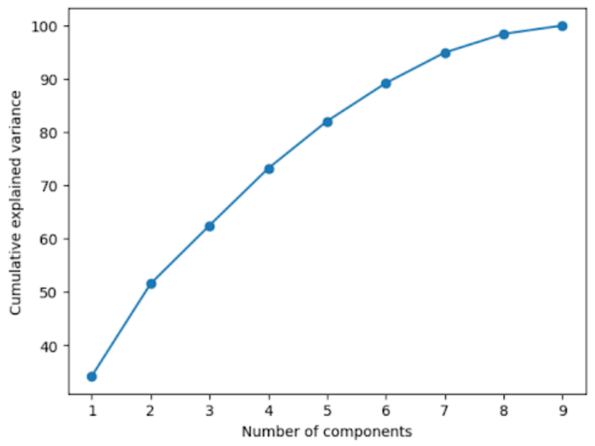

Figure 5. Principal Component Analysis

and their corresponding explained variance. The results were as follows:

- Explained variance: [34.16 17.35 10.94 10.73 8.87 7.16 5.72 3.51 1.57]
- Cumulative explained variance: [34.16 51.51 62.45 73.18 82.05 89.21 94.93 98.44 100.0]

If we consider at least 85-90% variance explained as a threshold, we found that it would require a projection onto a 6-dimensional space. However, this was not a considerable gain (reducing only 2-3 features) as compared to the possible information loss and reducing the accuracy of classification. Hence, we decided against further reduction of the selected features.

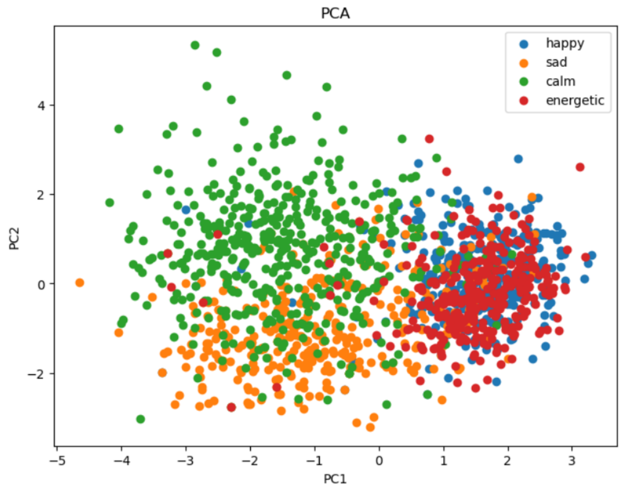

Figure 6. PCA Clustering

2.5. Classification:

2.5.1. Gaussian Naive Bayes

Why Naive Bayes?
We've implemented Gaussian Naive Bayes (GNB), a robust classification algorithm for our Spotify song data. With so many song characteristics (features) available, GNB performs well in scalability, effortlessly handling the high-dimensional nature of our dataset. This is particularly advantageous as our data points exhibit a Gaussian distribution, aligning with GNB's underlying assumption that the data follows a Normal distribution.

Our model capitalizes on the Gaussian assumption, utilizing the inherent distribution of our song features to enhance classification accuracy. Furthermore, the assumption of conditional independence among predictors fits well with our dataset, as we assume that different song features can influence the mood classification without depending on others. This allows the model to effectively classify despite the diverse range of features in our dataset, assigning equal importance to each characteristic in the final mood classification.

Implementation Details
We constructed the algorithm by implementing the underlying functions one by one. The steps involved are : We created a function that parses through the entire dataset, and separates the row according to their class value. Another function is implemented, which calculates the mean value and the standard deviation for each feature separately given a set of data points.

Using the above 2 functions, we first separated the entire dataset based on their class values, and then found the statistics (mean and standard deviation) for each feature separately for each class in our dataset.

Using the above statistics, we implemented a function that calculates the Gaussian Probability Distribution function for a given data point, using the formula :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{1}$$

Now, we tie together all the functions that we have implemented above to classify our datapoint into one of the 4 classes. For each of the 4 classes and for each of the 9 features, we use the Gaussian Distribution Function created using the mean and standard deviation for that class and feature, and calculate the probability that the feature x_i belongs to that class. We then multiply the 9 different probabilities found together, and multiply the value with the prior probability of that class. This final resultant value gives the probability of the data point belonging to that particular class. Repeating this procedure for all of the 4 classes, we find the class that maximizes this probability, and conclude the predicted class for that data point.

Experimentation and Evaluation We evaluated the performance of our model using the K-Fold Cross Validation technique. By setting the number of folds to 10, we observed the following results :

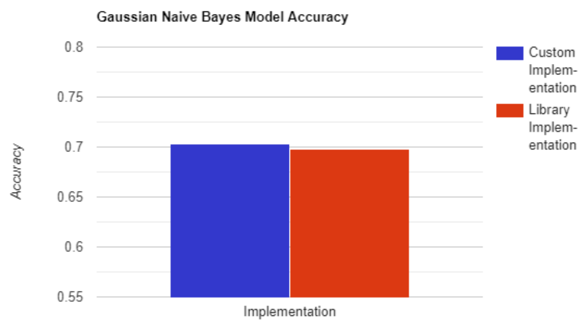

Figure 7. Naive Bayes Accuracy Comparison

Comparing with library implementation We compared the results with Sklearn Gaussian Naive Bayes model, which we also ran with 10 Fold Cross Validation

2.5.2. K-Nearest Neighbours Classifier

Why KNN?
We have used 4 models for our project, one of which is KNN, also known as K nearest neighbours. The K-nearest neighbors algorithm is a popular machine learning algorithm that is used for both classification and regression tasks. It works by finding the K-nearest data points to a given data point that we want to classify or predict the value of.
We chose to implement KNN because we have relatively few features, and KNN works well with less features. KNN is a non-parametric algorithm, which means it does not assume any specific form for the underlying data distribution. This flexibility is advantageous when dealing with diverse and potentially complex song attributes, as the algorithm adapts to the structure of the data without imposing restrictive assumptions.
KNN makes predictions based on the similarity of instances in the feature space. In the context of mood classification for Spotify songs, songs with similar features are likely to evoke similar moods. Therefore, KNN's ability to make localized decisions based on nearest neighbors is suitable for this task.

Implementation Details
Before applying this algorithm, we first need to calculate the distance between each data point that we wish to classify

with other data points. There are several distance metrics available, such as Euclidean distance, Manhattan distance, and Hamming distance. Euclidean distance is preferable for features with floating-point numbers, while Hamming distance works well for categorical features. Once we have calculated the distances between data points, we sort them in ascending order and select the first K neighbors. The value of K is a parameter that needs to be determined based on the minimum Root Mean Squared Error (RMSE) for the respective K. RMSE measures the differences between the predicted values and actual values.
After selecting the K-nearest neighbors, we determine the classes to which they belong and classify the data point to the class with the highest occurrence in the K-nearest neighbors.

Experimentation and Evaluation
In our experiment, we performed a evaluation of K-Nearest Neighbors (KNN) algorithm using 5-fold cross validation. This technique allowed us to split our dataset into 5 equal parts, where each part was used as a testing set once and as a training set 4 times. By doing so, we were able to estimate the performance of KNN with high accuracy and reduce the risk of overfitting.
Furthermore, we tested the KNN algorithm on different values of k that is num_neighbors, namely 5, 6, and 7. These values correspond to the number of nearest neighbors that KNN considers when making a prediction. By testing on multiple values of num_neighbors, we were able to find the optimal value that yields the best performance on our dataset. We have also performed paramater tuning, from which we can estimate the behaviour of model with the

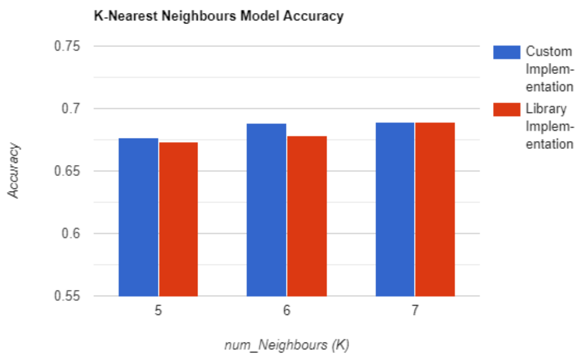

Figure 8. KNN Accuracy Comparison

Parameter Tuning
We did Parameter tuning for k = 1 to 21,and we obtained the following plot for Number of neighbors vs RMSE.

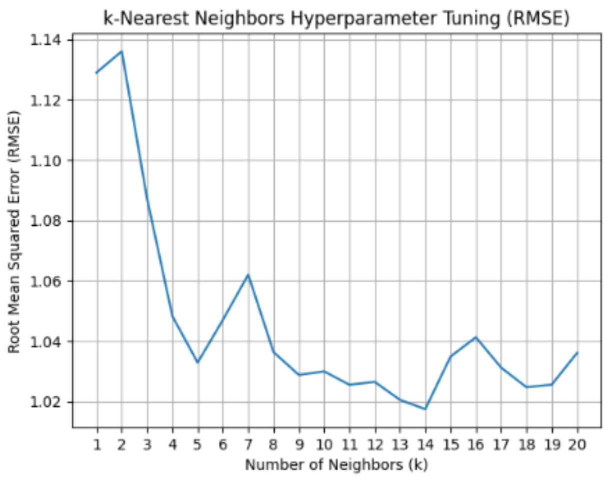

Figure 9. Parameter Tuning for KNN

From the figure it is visible that we get the global minima at k = 14.
Subsequently, for k = 14

- Cross validation scores: [70.61% , 68.92%, 67.23%, 73.31%, 69.59%]
- Mean Accuracy: 69.932%

Based on the plot of RMSE versus different values of k, it appears that the RMSE decreases as the number of neighbors (k) increases initially, and then it starts to increase again after reaching a minimum. The global minimum RMSE occurs at k = 14.
This trend suggests that with a smaller value of k, the model may suffer from overfitting, leading to higher RMSE due to capturing noise in the data. On the other hand, with a larger value of k, the model may become too generalized, resulting in higher bias and also higher RMSE.
Therefore, the optimal value of k appears to be around 14, where the RMSE is minimized. This value strikes a balance between capturing enough local information from the neighbors and avoiding overfitting to noise in the data. You can use k = 14 for your k-nearest neighbors model to achieve better predictive performance on unseen data.

2.5.3. Decision Tree & Random Forest

Why Random Forest?

We chose to implement Random Forest as one of the algorithms for classification since it is an ensemble learning method and we have a relatively small dataset(consisting of close to 1500 rows). Random forest is a good algorithm when there is a small dataset since it is a bagging of decision trees with Bootstrap. Each decision tree is fed with a sample of data with replacement, in this way even with a lack of data, there are better chances of making a good model that does not overfit the training data. Building upon the argument stated previously, it will also be robust to noise as it considers the results of multiple decision trees on different subsets of data before making the final decision, and hence it will be less sensitive to outliers which are very common while examining an artistic field like music wherein the audio features of a sad song like the tempo, loudness etc may match with that of a general happy song. Random Forest is likely to help us mitigate the problems caused by such kinds of outliers and result in a much more generalized model. Moreover, this can also model non-linear relations within features and mood labels and hence give better results considering that it is highly likely for non-linear relationships to be present as we have 9 features on whose basis classification is to be made and we have 4 classes in which each instance is to be classified.

Details of Implementation

To implement random forest, we have first implemented a Decision Tree class. The DecisionTree class is designed to construct a decision tree model capable of making predictions based on input features. We have tried 2 kinds of implementation.
First, which employs Gini Impurity to determine the optimal split at a node during tree construction and secondly which employs Entropy as a measure for the same task. Gini impurity is a measure of how often a randomly chosen element from the dataset would be incorrectly classified if it were labelled according to the distribution of class labels in the node. It ranges between 0 and 1, where 0 indicates perfect purity (all elements belong to a single class) and 1 indicates maximum impurity (the elements are evenly distributed across all classes).

On the other hand, entropy measures the level of disorder or uncertainty in a dataset's class distribution. It is calculated as the sum of the probabilities of each class multiplied by the logarithm of those probabilities, with a negative sign. In both cases, a lower value indicates higher purity and a more desirable split.

By iteratively evaluating potential splits across features and thresholds, the decision tree recursively partitions the data based on the split that minimizes the weighted sum of impurities in the resulting child nodes. This process continues until stopping criteria, such as reaching the maximum tree depth or achieving pure leaf nodes (nodes containing samples of only one class), are met.

Moreover, the implemented class also incorporates bootstrapping, a resampling technique, during training to enhance model robustness and diversity. By randomly sampling subsets of the training data with replacement, multiple bootstrap samples are generated, each potentially containing duplicate instances and omitting others. These samples are then used to train individual decision trees in the ensemble.

Bootstrapping facilitates the creation of distinct trees with varying training data, enabling the random forest model to generalize well to unseen data and mitigate overfitting. Within the RandomForest class, each decision tree is trained independently on a bootstrap sample of the original dataset, allowing for diverse sets of training instances and feature subsets to be considered during tree construction. During prediction, the RandomForest class aggregates the outputs of its constituent decision trees to reach a final prediction, often through a simple voting or averaging mechanism.

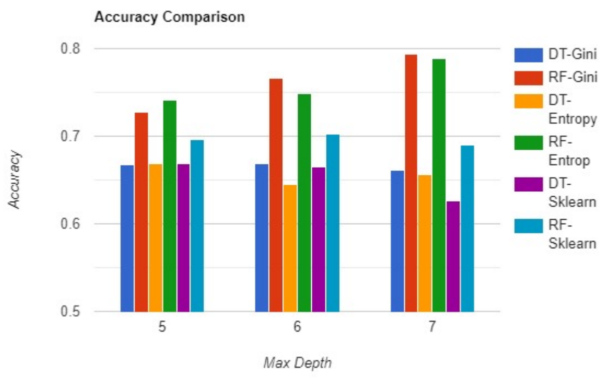

Figure 10. Random Forest Accuracy Comparison

Experimentation and Evaluation

We evaluated both Decision Tree and Random Forest models using 5 fold cross validation We ran the Decision Tree model on our data with different values of the parameter- max depth of the tree- we tried 5,6,7 as the values.

In the case of using Gini Impurity as a measure for determining the optimal split, we observed that the accuracy improved by increasing the depth from 5 to 6 while it decreased by further increasing the depth from 6 to 7. Hence, we can in a way say that 6 may be the optimal tree depth for our data. While evaluating random forest, we observed that the accuracy increased on increasing the depth from 5 to 6 as well as from 6 to 7.

In the case of using Entropy as a measure for determining the optimal split, we observed that the accuracy decreased on increasing the depth from 5 to 6 while it increased from 6 to 7. While evaluating random forest, we observed that the accuracy increased on increasing the depth from 5 to 6 as well as from 6 to 7.

For our data, Gini impurity turned out to give slightly better results than entropy, however, the difference is minimal only. Slightly better performance could be because we are using a relatively high number of features for classification, i.e., 9 so in those scenarios Gini impurity is better able to capture the information gain due to being less computationally expensive as compared to entropy measure. We did observe that the model with entropy took slightly longer to train because it used the logarithm function in the calculation which is computationally intensive.

For decision tree, we observed that our implementation is performing either better or almost similar to that of scikit learn implementation for the same set of parameters. For random forest, our implementation is performing better than the library implementation.

2.5.4. Conclusion for Classification Methods

After comparing the accuracy achieved by various classification methods, we find that Random Forest performs best for this task on our dataset, followed by Naive Bayes, KNN and then finally Decision Tree. They are compared using accuracy as the metric.

2.6. Clustering:

K-Means Clustering

While doing EDA and observing the data as well as while attempting to scrape our own dataset using the Spotipy library and analyzing the data that we were able to scrape, we found out that there is a lot of overlap of songs between different mood labels which makes sense as well because music is a very subjective field in the sense that what one deems to belong to a calm category might be in the sad category for another person. Moreover, since here we are only using audio features such as tempo, loudness, acoustics, etc and not the lyrics of the song, we will not be able to appropriately label the songs.

Hence, we have also explored unsupervised learning for this task. In this case, the problem can be formulated as grouping similar songs(with respect to the audio features we have) together so that if a user likes a song, he/she can then access similar songs to that one and this can greatly help in increasing user satisfaction as most people define song similarity as the similarity of audio features only.

We have used K-means for clustering as firstly it is an easy algorithm to implement which is guaranteed to converge. It is also pretty fast when compared with other clustering algorithms. Moreover, it is evidenced that K-means performs well when we have an idea about the number of clusters that will be formed, which in our is known and will be around 4 only.

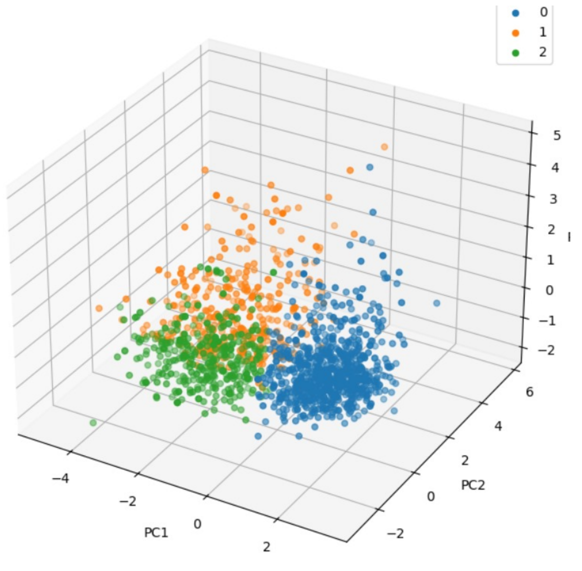

Figure 11. K-Means Clustering (Custom Implementation)

Implementation

Initially, K centroids are randomly selected from the dataset. Then, in an iterative process, each data point is assigned to the nearest centroid based on Euclidean distance. After all points are assigned, the centroids are recalculated as the mean of the points assigned to them. This assignment and centroid update process repeats until convergence, typically defined by minimal movement of centroids or reaching a maximum number of iterations. The result is K clusters, each characterized by its centroid, with data points grouped according to their proximity to these centroids.

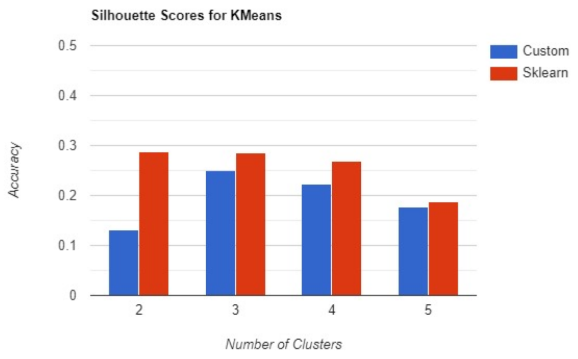

Figure 13. K-Means Silhouette Score

Model Evaluation and analysis on parameter tuning

We have used Silhouette score to evaluate the quality of

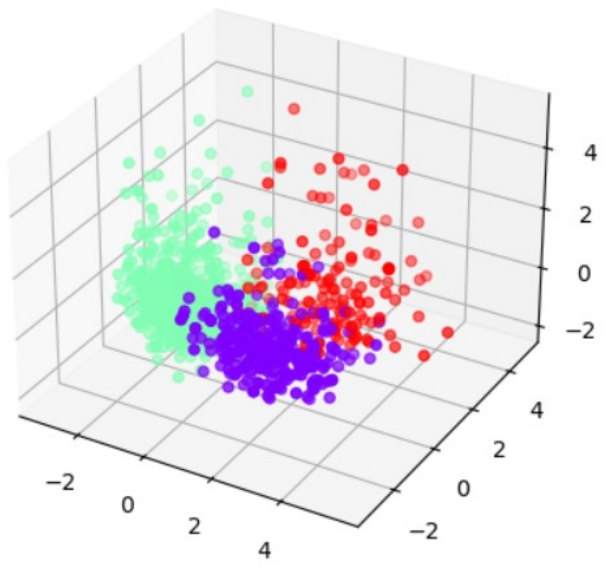

Figure 12. K-Means Clustering (sklearn)

clusters generated. This score is the mean Silhouette Coefficient of all samples. The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is $(b - a)/max(a, b)$. To clarify, b is the distance between a sample and the nearest cluster that the sample is not a part of. The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters.

We can see that the score increased from k=2 to k=3 and then decreased on increasing k further. Hence, we can say that k=3 is the optimal number of clusters for our data. This validates the analysis given below as to how happy and energetic songs are very similar and have mostly been grouped in the same cluster while sad and calm songs tend to form their separate clusters while some sad and calm songs are grouped together in the last cluster.

Sklearn's implementation is performing better than our custom implementation on all values of k. It is also performing best for k=3, similar to our implementation. Its better performance may be attributed to what centroids are chosen randomly in the initial step.

3. Challenges Faced

- Scraping of data took a lot of time due to the rate-limiting of the API. It allowed only 100 songs to be scraped at a time. Even after the time spent gathering data, it didn't turn out well due to the overlapping of songs and different preferences of people while making mood playlists.
- The computational time required to test the efficacy of our implemented models, such as random forest and K-Means, was quite substantial. We had to endure a lengthy waiting period every time we tested the models to ensure their accuracy or to detect any potential errors.
- The K-Nearest Neighbor (KNN) model took a considerable amount of time for parameter tuning.
- During the training of the random forest model, we encountered 'out of bounds' errors on the leaves. Due to this issue, the training process took a considerable amount of time, and we had to wait for a significant period every time the error appeared.

Notebook link for reference:

- CS361 ML Project

4. Conclusion

We observed how many points from each mood class belonged to each cluster. It showed stark similarities between the happy and energetic songs. Some of the calm songs were clustered separately, while the others overlapped with the cluster majorly consisting of sad songs. This was even more clearly visible when plotted into a 3-dimensional space (using PCA). This is in coherence with our understanding of music where the characteristics of happy and energetic songs are similar- Loud, lively, and danceable, while the calm and sad songs have similar characteristics- slow, acoustic, and sedate.

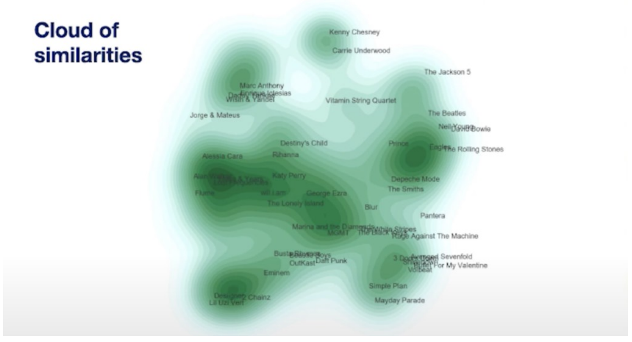

Figure 14. Spotify Clustering by Similar Artists

This is an important observation as it highlights the importance of unsupervised learning for this use case which mostly consists of song recommendation and classification based on the mood. It is a highly subjective matter, where it is best to index the songs based on similarity rather than some pre-defined classes such as mood. This can also be seen in the song recommendation system of Spotify itself, which recommends similar songs across genres and languages.

On learning more about the Spotify recommendation, we discovered that it also depends on clustering songs based on different types of similarities but on a diverse set of features, more than what is available to us through the public API-

- User-entity affinity: ”How much does user X like artist A or track B? What are the favourite artists/tracks of user Y?”
- Item similarity: ”How similar are artist A & artist B? What are the 10 tracks most similar to track C?”
- Item clustering: ”How would we split these 50 tracks/artists into separate groups?”

The main aim of our mood classification was for better recommendations based on the mood of a person. This study has led us to the conclusion that the similarity of audio features to determine the mood is a better parameter of recommendation, rather than labelling songs into different moods. Clustering on multiple parameters performs better for the objective than any classification algorithm for our purpose as people tend to focus more on the audio features of any 2 songs while determining similarity thus, making it a very subjective task. Classification here is very hard bound in the sense that a single instance can belong to only a single class out of the available mood classes.

References

Moschitto, M. *GitHub Dataset*, 2022.

Nuzzolo, M. *Music Mood Classification (Medium Article)*, 2015.

Patra, B. G., Bandyopadhyay, S., and Das, D. *Unsupervised Approach to Hindi Music Mood Classification*, 2013.

Solano, H. D. *Clustering My Spotify Songs and My Mood Prediction*, 2022.

Wallach, J., Corr, B., and Moschitto, M. *Mood Classification of Spotify Songs*, 2021.