Data Mining Exam Project

Simon Min Olafsson, Maja Styrk Andersen,
Emil Højrup Vinther, Malte Helin Johnsen

I. Introduction

This project was written in November and December of 2023 at ITU as the final exam project for the course Data Mining. The full data set and source code for the project is available in the public GitHub repository. To ease navigation, we will highlight relevant source code in the beginning of each section. All source code is available as a regular Python file as well as a Jupyter notebook with plots and output.

A. The Data Set

Throughout this project, we have been working on the data set *"30000 Spotify Songs"* which was retrieved by the authors in January 2020 [1]. The data set is retrieved with the package spotifyr which uses the Spotify Web API [2] and is simply stored as .csv file in the file `spotify_songs.csv`. The spotifyr package is authored by Charlie Thompson, Josiah Parry, Donal Phipps, and Tom Wolff. This package allows users to retrieve either their own data from Spotify and as an example query one's own most recently played track. We chose this specific data set as all group members can relate to music and use Spotify in our every day life. Furthermore, we were curious on how a recommender system can look like when dealing with rather personal attributes such as taste and musical preferences. One group member has some domain knowledge about music theory, which enhanced our ability to understand the more music-technical aspects of the data set.

B. Work Distribution

In agreement across the group we state that everyone contributed equally and with fairness to the exam project. We met weekly to discuss tasks at hand, and set the goals for what to achieve in the given week. We found that some sections of a data mining project are meaningful to conduct on plenary sessions such as Exploratory Data Analysis. We also found that experimenting with machine learning models is easier to conduct individually. By that, we individually experimented with model creation ourselves, and reported back to the group when major findings was established, and in the end evaluated these findings. More specifically, we carried out the EDA together, with Emil doing the majority of the work. Simon and Emil worked on supervised models, with Simon working on the Random Forest algorithm for predicting danceability and Emil working on Decision Tree classifiers for predicting song genre. Malte and Maja experimented with unsupervised models, where both of them applied various clustering algorithms for two different tasks, with Maja working on a song recommendation system based on mood and Malte working on an autoplay song recommendation.

II. Exploratory Data Analysis

To get an understanding of the possibilities and limitations of the data set and how the different features interrelate we have used several different plotting techniques some of which we will detail in the following section. The relationship between data mining and exploratory data analysis (EDA) is an iterative approach where results of the EDA informs what models you choose for machine learning and the findings from these models might then impact what further EDA you undertake. Most of the EDA we did is found in the `eda.py` file but the file `kmeans_eda.py` also contain some EDA. We also use plots to explore and analyze the results of most of our models to get an understanding of their performance and how this relates to the data.

A. Data Reliability

Relevant source code:
`eda.ipynb`
`kmeans_eda.ipynb`

The data set is collected from the public Spotify API [2] but contains no explanation on the inclusion criteria for the songs that were picked. Simple analysis shows that all songs are from one of the following genres: *EDM, latin, pop, r&b, rap, rock*. This means that many forms of music are left out like classical music, jazz and non-western styles of music. This is not necessarily a problem but worth keeping in mind when designing a recommendation system since it limits the musical variety such a system is able to recommend. We mostly focus on the songs' audio features. We do not know how these were calculated by Spotify since the algorithms they use are proprietary, so we set out to explore some edge cases based on some musical background knowledge.

We used the function print_min_max_features() to investigate the most extreme songs for each feature and listening to them. This led to some insights: the least danceable song is in fact not a song but a rain forest recording confusingly labelled under the genre *tropical latin*. This "song" also has a tempo, key and mode despite these features making no sense for a field recording. Similarly the most energetic song was a recording of waves also labelled tropical latin. Furthermore, the data set includes the features "key" and "mode" but "mode" only has two possible values: major and minor effectively making it impossible to correctly categorise songs that are in alternative modes like the rest of the church tonalities, jazz and blues scales and non-western scales. An example is the song "Army of me" by Björk. This songs is in c locrian but classified as f# major in the data set.

This is, admittedly, an unusual song but still points to the limitations of this categorisation. It is also not possible to capture modulations which is widely used in especially pop music.

B. Preprocessing

Relevant source code:

`utils.py`

By conducting initial EDA and analysing data reliability we learned that the data set is rather clean. Those extremes that exist are not feasible to correct or exclude, considering the manual labor and knowledge of the specific songs it would require. We acknowledge the presence of the above mentioned issues and recommend the reader to keep these in mind throughout the conducted research.

The data set initially contained 32,833 entries. Upon closer inspection, we found that within these entries only 28,356 values of *'track_id'* were unique. This is due to the fact that the same song appears on multiple playlists and therefore also has multiple entries. All songs have a *genre* and a *subgenre* label from the playlist that they were retrieved from. This also means that the same song can be labelled with a different *genre* for each entry (e.g. the song "HUMBLE" by Kendrick Lamar is both categorised as pop and rap). We chose to handle this issue by removing all duplicate occurrences of *'track_id'* in the data, simply keeping the first occurrence. Furthermore, in the preprocessing step we split the data into a training- and a test set, ensuring that the test set is the same for each model. We leave it to the individual model training to further split the training set to include a validation set. Before training each model, we conduct further preprocessing on the data, such as feature selection, to fit the purpose of the specific model. This is specified when describing the models throughout the report.

C. Relationship Between Features

Relevant source code:

`eda.ipynb`

We created different plots for investigating the relationships between the audio features. First we made a few pairplots using the Seaborn Python package, but weren't able to extract much useful information from them. We then made a heatmap plotting the correlation matrix for the various features in figure 1. This only led to some superficially interesting insights: *energy* and *loudness* are positively correlated while *energy* and *acousticness* are negatively correlated. None of which are very surprising.

We plotted each of the numerical features to understand their value distribution. We find that each of them follows a relatively uni-modal normal distribution. More useful insights were gained from making boxplots for each feature grouped by genre. One example is the boxplot for danceability (see figure 2) which shows the *rock* genre to deviate from the other genres suggesting danceability to be a useful feature for categorising rock songs. The notebook `eda.ipynb` contains some further plots that we found interesting but did not use for our further investigation.

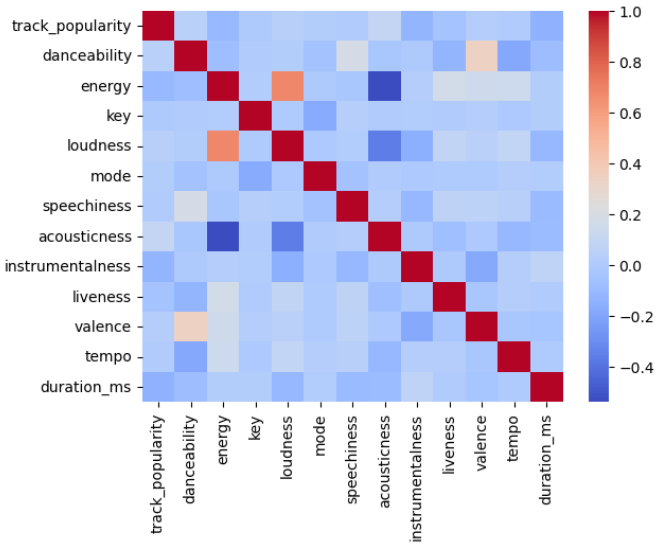

Fig. 1. Correlation matrix for audio features

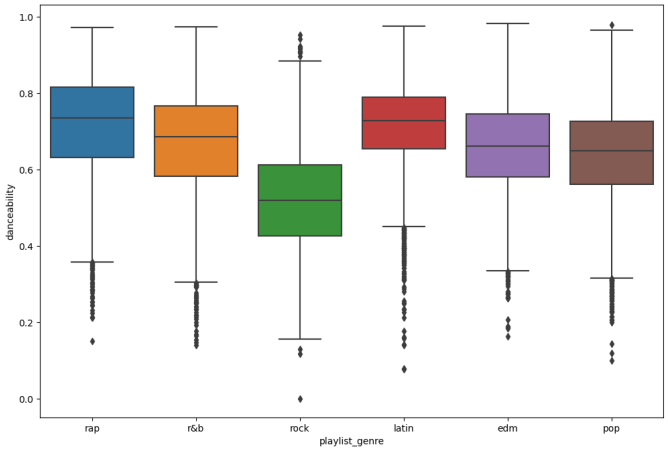

Fig. 2. Boxplot for danceability by genre

III. Models for Attribute Prediction

A. Predicting Danceability with Random Forest Regression Model

Research question:

- Can we predict the danceability of song by using audio-related features only?

Relevant source code:

`danceability.ipynb`

To predict the danceability of a song, we first analyzed which features were relevant to define danceability. In our correlation matrix (see figure 1) we could see, that no single feature had a strong linear relationship with danceability. The only feature that had a semi-strong linear relationship with

danceability was valence with a correlation coefficient of roughly 0.35, where valence describes the musical positivity conveyed by a song. Due to the shown fact that there is not a clear correlation between danceability and the other features, we decided to use the Random Forest Regressor as it does not assume a linear relationship unlike for example Linear Regression, while also comparing the performance to other regression models. When running our model on all numeric audio-related features, we saw that 'tempo', 'speechiness', 'energy' and 'valence' had the highest feature importance (see figure 4) and conveyed danceability more than duration and liveness of a song for example. We also excluded features with categorical values, as we want to predict on continuous data which is required for regression tasks.

The Random Forest algorithm is a supervised learning algorithm, that can be utilized as both a classification and regression model. The algorithm is based on ensemble learning, which is a process of combining multiple decision trees, using the most common label across all trees to improve the prediction accuracy of a given data set. Using Random Forest Regression, it calculates the mean of all the predictions made by all individual trees, where the final prediction can be enclosed as the average of predictions made from all the trees in the forest. This also makes the algorithm robust towards outliers, as they get averaged out when aggregating the multiple tree predictions. With this aggregation, the algorithm should also be less prone to overfitting with the ensemble approach.

In addition to evaluating our models performance, we will also investigate each features impact on the prediction result by extracting feature importances from the RF model. In a single decision tree when a split occurs using a specific feature, the aim is to maximize the reduction in impurity. The calculation of a feature's importance is based on how much each feature contributes in reducing this impurity (we use gini impurity?) at each node, which is also weighted by the probabilty of reaching that node. The feature importances scores are normalized as they sum up to 1, which also allows relative comparison between the features. When extracting a feature's importance in a random forest model, the sum of every features importance value across all trees are then divided with the total number of tress.

We used several hyperparameters to tune our model, such as the number of estimators (trees), the max depth and the number of features to consider when splitting at each node. When looking at the results, the feature 'tempo' ranks higher in feature importance (see figure 4) compared to 'valence' despite the latter having a higher correlation coefficient. This might be due to the RF model capturing non-linear relationships and complex patterns between 'tempo' and 'danceability', which signifies that 'tempo' plays a significant role in predicting danceability of a track. When running our regression task with the hyperparameter tuning that were considered best, we achieved a MSE of 0.012 and MAE of 0.087 running on the validation set, and a MSE of 0.014 and MAE of 0.093 on the test set, which is to be considered

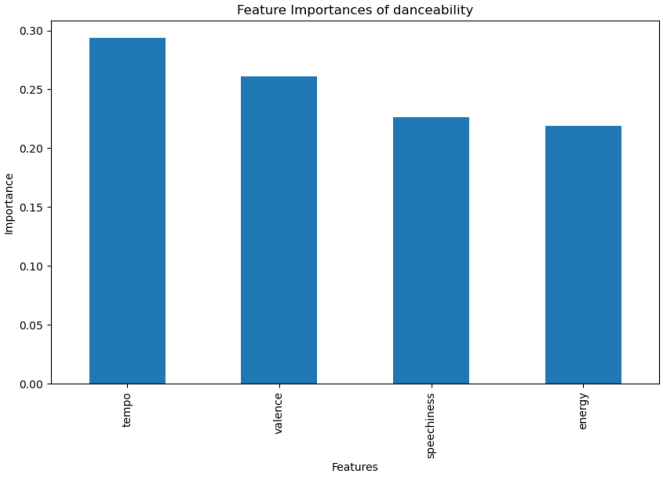

Fig. 3. Most importanced features of danceability

satisfactory. However, we achieved an R^2 score of 0.41 in the validation set and 0.35 in the test set, which is below moderate of explaining the variance in the data. This means that the model could be affected by factors like underfitting (too simple model), or a insufficient number of features, as there could be more features that are not present in the dataset, which may influence danceability.

B. Predicting Song Genres

Research question:

- Can we accurately predict the genre and subgenre of a song using a supervised model?

Relevant source code:

`genre_classifier.ipynb`

To address this question we first wanted to test whether the audio features were in fact useful for predicting the genre and subgenre labels. This is not necessarily the case. Features that would more obviously describe the differences between genres such as song structure (verse, chorus etc), instrumentation (synthesizers, guitar, horns etc), melodic complexity or lyrics are not included in the data set. After experimenting with different models we decided on using a decision tree classifier for predicting genre labels. The decision tree algorithm recursively splits the data set creating a tree structure where each internal node represents a test on an attribute. An example could be the node "is the energy ¿ 0.7" which leads to a branch with energy below and one equal to or above 0.7. The best structure of the tree (the ordering of which nodes come first) is calculated using a metric such as information gain or Gini impurity. The Gini impurity is a measure between 0 and 1 of how pure the subsets are. 0 means perfect purity (all instances in the subset belong to the same category) and 1 means complete impurity (instances are evenly spread across categories). The decision tree algorithm minimizes the weighted sum of Gini impurities.

We first created a model by training on all audio features and using the default parameters for sci-kit learn's DecisionTreeClassifier. When developing the model we split the data into a train and a validation set. For reporting the final performance metrics we then used the test set created in the `get_dfs()` function.

The model had decent performance but when inspecting the importances the model gives to each feature (by plotting the model's inbuilt feature_importances_ attribute) we found the features 'key' and 'mode' to be the least important 5. Since these features are also not entirely reliable, as described above, we decided to discard them. We created a better performing model by using grid search cross validation hyperparameter tuning. This led to better performance with the following parameters: 'criterion': 'gini', 'max_depth': 10, 'min_samples_split': 8, 'splitter': 'best'. [1]

To further improve the performance of the model we also experimented with using the `track_name` and `track_album_name` features. This required us to turn the text data into a useful format for the Decision Tree algorithm. For this we used the TFIDF (term frequency inverse document frequency) vectorizer. The term frequency part calculates how often a term occurs in a document and the inverse document frequency part measures the importance of a term across a collection of documents. It transforms the text data into a vector with a score for the terms in the text in relation to each document where a high score means the term is both frequent in that document and relatively rare across the entire corpus.

We first created a model only using the text data which performed rather poorly. But combining the audio features with the text data resulted in the best performing model for predicting genre labels. The model is fitted using a pipeline with a preprocessor column transformer and the Decision Tree model. The preprocessor transforms the text data to a vector using the TFIDF vectorizer and scales the audio features using a Standard Scaler. The classification matrix for the model can be seen in figure 4. For the final evaluation we used the test data set created in `utils.py`. We also compared the models to a baseline dummy classifier finding that all the models performed better than the dummy model. When evaluating the performance of the model it is also worth keeping in mind that a song can belong to several different genres so we do not expect 100% accuracy. [2]

This also led us to the conclusion that at least most of the audio features combined with song and album title are indeed useful for grouping the songs into genres. Whether this is the best way of grouping the songs is not clear from this analysis alone but will be investigated further below.

[1]We also experimented with predicting the *subgenre* attribute and succeeded in doing so relatively accurately. These results can be seen in `genre_classifier.ipynb`

[2]The plots for the models not included here (such as the baseline dummy model) can be seen in the file `genre_classifier.ipynb`

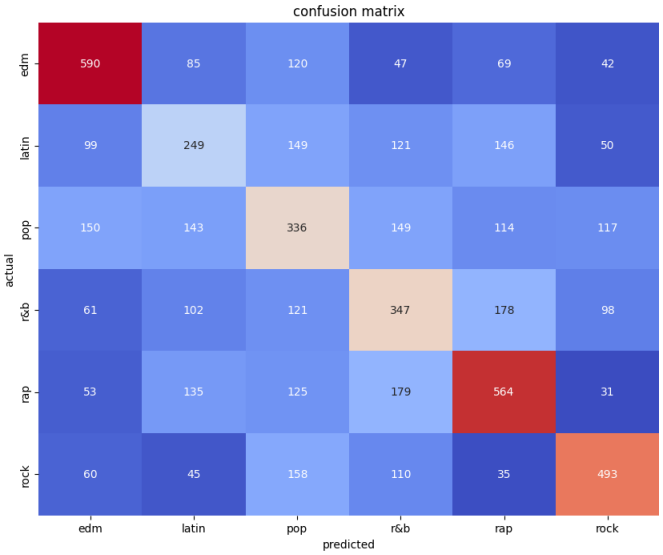

Fig. 4. Confusion matrix for Decision Tree model for predicting genre labels using both text and audio feature data.

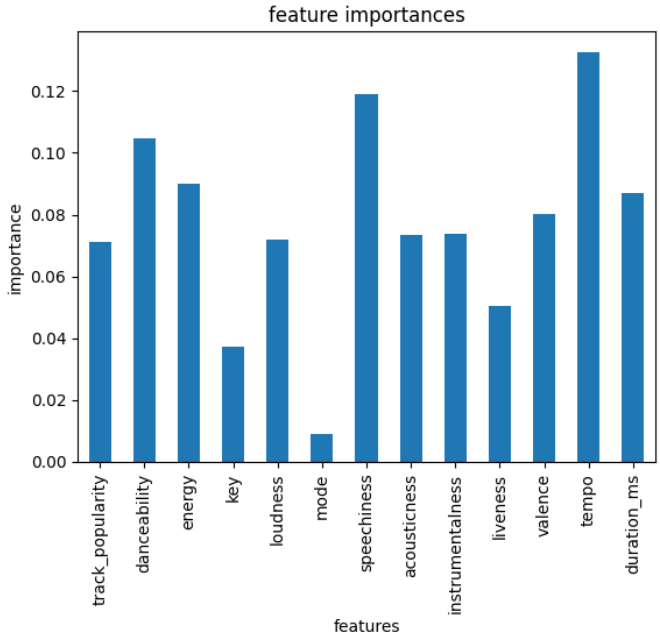

Fig. 5. Feature importances for Decision tree model for predicting genre labels

IV. Music Recommendation Models

A. Recommendation Based on Listener Mood

Research question:

- Can we cluster the data set based on mood-related features to provide music recommendation?

Relevant source code:

`mood.ipynb`
`OPTICS.ipynb`

```
Evaluating Feature Combination: ('valence', 'danceability', 'loudness', 'tempo')

  Evaluating Number of Clusters: 2
/opt/anaconda3/envs/spotify/lib/python3.12/site-packages/sklearn/cluster/_kmeans.py:14
  super()._check_params_vs_input(X, default_n_init=10)
    Silhouette Score: 0.5386001041938763
```

Fig. 6. Example of iteration

K-means: The goal is to feed the machine with a mood, e.g. "needing an energy boost" and then get recommended a song from the cluster providing energy boosting songs. We applied the k-means algorithm, which is described in section B further below. One of the main disadvantages of the K-means algorithm is, that the user must determine how many clusters the data holds before applying the algorithm. Therefore, an iterative approach to testing feature combinations and number of clusters can eliminate this disadvantage. To find the best number of clusters we iterated, grid-search alike, over a range of clusters and a set of features. The set of features were hand-picked and are features which we believed to have an influence of human moods – however, we are aware that taste is a subjective matter. The chosen features are *energy, valence, danceability, loudness,* and *tempo*.

Before iterating over the ranges of clusters and feature combinations we standardized the data. We apply `StandardScaler` from the SKLearn module, which scale the feature values such that the mean value is 0 and values are expressed as a factor of the standard deviation. This is chosen rather than `MinMaxScaler`, since the `StandardScaler` is more resistant to the presence of outliers and preserves the distribution of values more accurately. An example of an iteration is shown in the terminal output snippet in figure 6 where the combination *valence, danceability, loudness,* and *tempo* is used to create two clusters, leading to a silhouette score of 0.53.

After the iterations, we found that the best combination of features is *energy, valence, danceability,* and *tempo* and the best number of clusters is three. It returns a silhouette score of 0.62. Thereby, the feature *loudness* is dropped which upon further reflection makes sense since this is the one feature out of the five initially chosen features that is least correlated with mood. While this is not ideal, but also not too bad, we plotted the resulting clusters in 3D space to investigate further.

By visual inspection of the 3D plot in figure 7 we see that it is challenging to conclude whether the resulting plot adheres to a notion of good clustering: high intra-class similarity and low inter-class similarity [3]. Other options to find number of clusters include the elbow method. However, the grid search-inspired method was interesting to apply since we were able to follow along with which features was tested, what combination of features, and lastly, what silhouette score the different combinations obtained.

We investigated the given clusters a bit further with descriptive statistics and another plot. Figure 8 plots the clusters, their features, and their mean scores between 0 and 1. In cluster 0, *energy* scores 0.72, *valence* scores 0.70, and *danceability* scores 0.71. In cluster 1, *energy* scores 0.50,

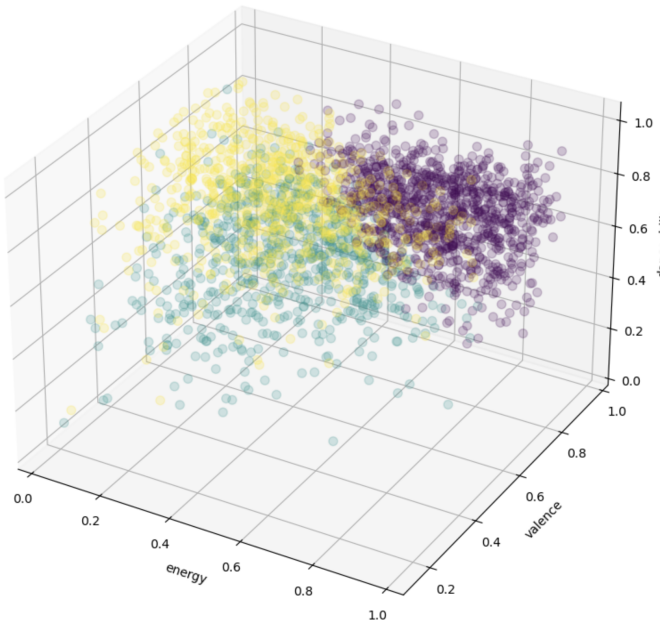

Fig. 7. x: energy, y: valence, z: danceability in original scale

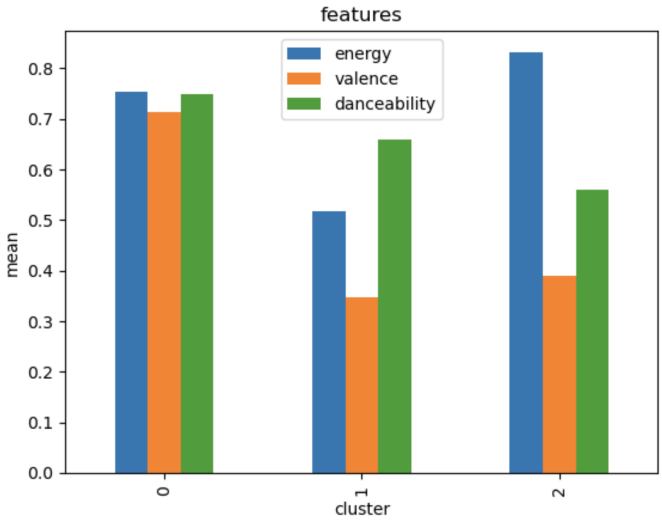

Fig. 8. Histogram of the mean values of the different features in each cluster

valence scores 0.35, and *danceability* scores 0.68. In cluster 2, *energy* scores 0.82, *valence* scores 0.40, and *danceability* scores 0.57. Applying domain knowledge, we define cluster 0 as "needing an energy boost" since it has a high score on all features and thereby we believe it will create a high impact on mood. We define cluster 1 as the mood "feeling rhythmic" since danceability scores high, and is the most dominating feature. We define cluster 2 as "feeling happy". We are aware that this subjective labeling of the clusters can introduce bias to the recommender.

In figure 9 descriptive statistics upon the clusters can be seen. Statistics include the mean, median, and standard

	energy			valence			danceability		
	mean	median	std	mean	median	std	mean	median	std
cluster									
0	0.742202	0.753	0.129809	0.708770	0.714	0.148012	0.744616	0.748	0.094432
1	0.506081	0.518	0.151529	0.350861	0.346	0.163755	0.645592	0.660	0.143046
2	0.808266	0.832	0.130968	0.399050	0.389	0.190921	0.547278	0.559	0.123522

Fig. 9. Descriptive statistics of the clusters

deviation for each cluster and each their feature. In general, the standard deviation score is relatively low which per se is a good sign, suggesting higher cluster density. Having the labeled clusters in mind from above, the descriptive statistics to some extend confirm that e.g., cluster 0 align with the mood "needing an energy boost" as it has a high mean and median score in all the features.

Ideas for further studies include feature engineering. Via discretization of the data set, it would be interesting to determine the clusters based on e.g., their impact by multiplying energy and loudness. One might suspect that the clustering can look different including such an engineered feature. Furthermore, user testing, e.g., as in active learning would be ideal. In active learning, a user with expert domain knowledge would rate how well a given recommended song align with the label it has retrieved.

OPTICS: We tried another clustering algorithm that better captures non-spherical shaped clusters to see if this can provide a better clustering [3]. The OPTICS algorithm can identify arbitrarily shaped clusters. This density-based technique is based on the idea to continue growing a cluster until the neighborhood is not dense with data points anymore [3]. In theory, a major advantage of the algorithm is that the user does not need to pass it a specific number of clusters beforehand as is the case with k-means. One of the main disadvantages of the OPTICS is the time complexity being $O(n^2)$. However, since our data set is relatively small we deem it as an experiment worth trying. Another point of awareness is, that OPTICS is better for high-dimensional data which our data is not.

Applying the OPTICS algorithm, hyper parameters are set to configure the requirement of a minimum number of samples for a neighborhood to be considered dense. A core distance parameter is set as well, which is the minimum value of radius required to determine if a certain data point is a core point in a cluster. Reachability Distance is plotted based on reachability of the data samples to the core data point in each cluster. Plotting the reachability, we can interpret groups of data points, after having been assigned to a cluster, which has the distance minimum required distance to a core point. Thereby, if clusters are present, the reachability plot will form x number of valleys which signifies x clusters.

Figure 10 shows the reachability plot on our data set. The horizontal line determines a threshold for when the points can be considered densely packed, and thereby part of a given cluster. In the plot no valleys are formed - more specifically one large valley is formed. This means, the algorithm detected only one cluster, which all data points belong to. With other words, one large cluster signifies that there are no clusters of

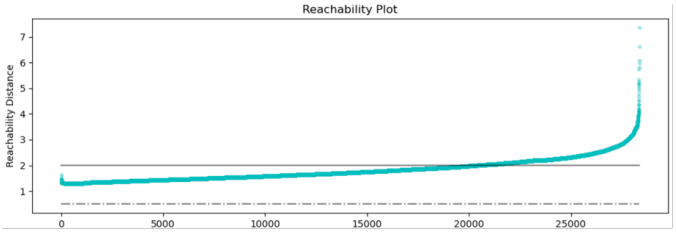

Fig. 10. Reachability plot shows only one big valley

arbitrary shapes present in the data set. We experimented with different hyper parameter tuning, however they all resulted in the same type of plot as seen in the figure.

B. Song Recommendation Based on Clustering

a) Research Questions

- How are the songs best clustered based on their audio features?
- How can the clustering be used for music recommendation?

Relevant source code:

`song_recommendation.ipynb`
`song_recommendation.ipynb`

Feature selection: For clustering the songs with a final goal of providing a tool for music recommendation, we reduce the attributes of the data to be clustered, to only contain numerical features related to the sound of the song. We choose to cluster based on these features because we want to recommend similar sounding songs. Therefore we first of all limit the data to the following audio-related features: *danceability, energy, speechiness, acousticness, instrumentalness, liveness, valence, tempo.* As previously mentioned, we are aware that these features can only explain the "sound" of a song to a very limited degree as they do not include a lot of important information such as the instrumentation.

Clustering tendency: We then want to examine if it is even possible and meaningful to cluster our data according these features. To do so, we have implemented a method for calculating the Hopkins Statistic (`hopkins_statistic(X)`) of a given data set. The intuition behind the Hopkins Statistic, is to examine how close our data is to a uniform random distribution in the same data space using the nearest-neighbor distance as the metric of comparison. The output of the method is a value between 0 and 1, where 0.5 indicates uniformly distributed data and 0.0 indicates non-uniformity as defined in [3]. We find that we achieve a result of roughly 0.0775 for non-scaled feature values, This is a very acceptable result, suggesting that it is a possible to identify meaningful clusters in our data. Due to the nature of the clustering algorithms we will use, it is also important to determine the Hopkins Statistic for the scaled feature-space. When scaling the feature values with the `StandardScaler` we achieve a slightly worse but still acceptable score of 0.152.

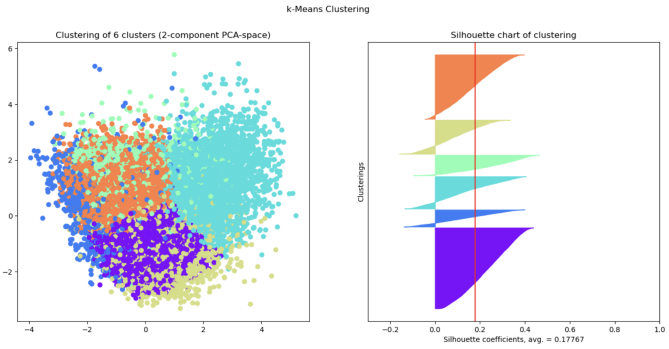

Fig. 11. Plot of PCA-transformed, clustered data and Silhouette Chart

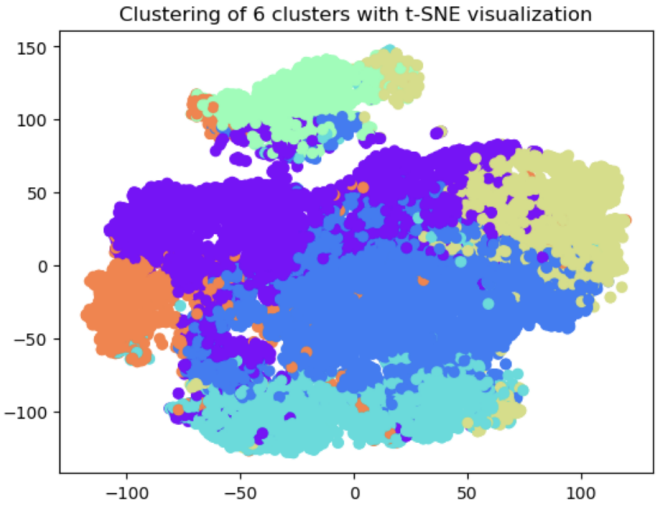

Fig. 12. Best achieved clustering (K-means) transformed with t-SNE

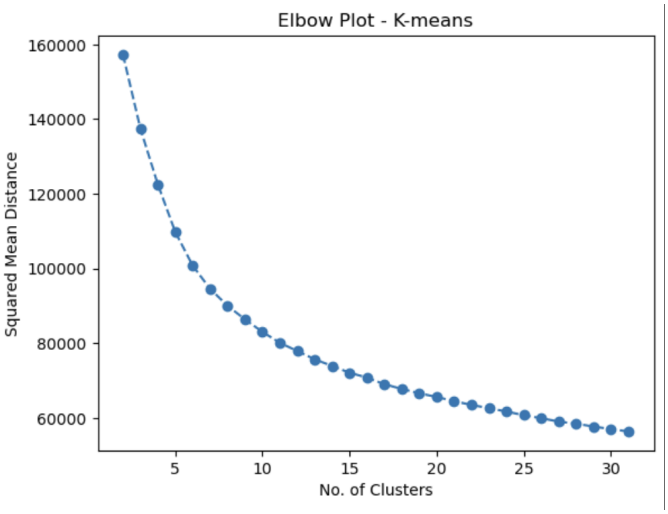

Fig. 13. Elbow plot of K-means clusterings

Clustering method: To define a good clustering of our data, we run the K-means clustering algorithm in the range 2-32 clusters on the scaled data. Scaling the features to be clustered is an important preprocessing step for the K-means algorithm, because the algorithm is attribute scale sensitive, meaning larger-scaled attributes such as *tempo* will dominate attributes on a smaller scale (*acousticness*) and therefore skew the clustering based on that feature. K-means works by partitioning points according to their distance to a set of K initial points. Based on this partitioning, a centroid for each partition is calculated and points are again partitioned based on their distance to the the centroids. This process is repeated until clusters converge. We use the 'kmeans++' initialization of the algorithm, meaning that the initial partitions are based on probabilistically determined centroids, rather than centroids drawn at random from the feature-space or sampled from the data.

Best clustering: Each clustering is plotted by first transforming the data to fit the 2-dimensional space using Principal Component Analysis. Note that the clustering representation in the 2-dimensional space does not provide much information about the quality of the clustering in itself, considering that the 2 components cover only roughly 40% of the explained variance. Alongside each PCA-plot, we also provide the Silhouette plot in order to pick the best number of clusters in the range. The Silhouette plot essentially describes for each point, how well it fits within its own cluster, compared to the 'best' other cluster, using mean intra-cluster distance and minimum mean inter-cluster distance as the metric for fit. A negative coefficient means that the point has a lower mean distance to at least one other cluster than its own, and therefore should most likely not be in the cluster it is in. Using this method covering a sample of 10% of the data, we find that the number of clusters that yields the highest average silhouette coefficient is 6. For this clustering we achieve a relatively low score of 0.178 and we also notice that this clustering shows a few negative silhouette coefficients. This however turns out to be the best clustering we can achieve on the feature-set, seeing that we get worse results for both Hierarchal Agglomerative Clustering (Ward Linkage, Single Linkage) as well as a Gaussian Mixture Model using the Expectation-maximization Algorithm and evaluating the clustering achieved by each of these in the same way. We also plot the Elbow-chart for the clustering range, using the squared mean distance to cluster centroids in order to validate the number of clusters. This method only proved mildly helpful on the clusters as only a very, very slight elbow-point can be identified at 6 clusters, even when plotting smaller cluster intervals.

Using t-SNE (t-distributed Stochastic Neighbor Embedding) we can plot this specific clustering in a more meaningful way than the PCA-plot, exactly because t-SNE aims to create a faithful 2-d representation of data in higher dimension. We configured the t-SNE algorithm for 3.000 iterations and a perplexity of 50 using a simply trial and error approach. The resulting representation seems relatively well-formed. Visually, this clustering also seems to be relatively well-fitted with a some obvious exceptions. We can notice that the same cluster (light-green/yellow) that shows a larger number of

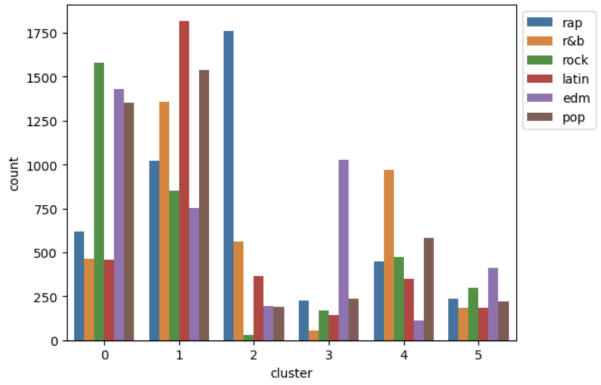

Fig. 14. Genre distribution per cluster

negative coefficients in the silhouette chart, also shows up as a more distributed cluster in the t-SNE plot. However, we should be careful to conclude to much about the clustering on the t-SNE plot due to the immensely different results it can produce based on hyperparameter configuration.

Genre distribution: Considering that the data set comprises music from exactly 6 main genres, we found it worthwhile to investigate if our clustering method had essentially just identified these 6 genres through their audio features. However, plotting the genre distribution for each cluster, we find that this is not the case. Most clusters have a relatively even distribution of genres, with the main two exceptions being cluster 2 and cluster 3, which lean quite heavily towards the genres 'rap' and 'edm' respectively. Even though cluster 2 is dominated by songs from the genre 'rap', we also find that there are more songs from the genre belonging to other clusters than cluster 2. This is also true for cluster 3 and the 'edm' genre.

Music recommendation: The clustering that we have created using the K-means algorithm can be used for music recommendation in multiple ways. The most straight forward method is to recommend/queue the next song to be played based on the song you are currently listening to. This is akin to the 'Autoplay' feature in Spotify. One way to implement this feature based on the clustering, would be to classify the song that is currently playing to one of the clusters and either at random pick a song from the same cluster or using certain weights to pick a more qualified song from the cluster, such as picking one of the more popular songs. A simple implementation of this method is implemented in the file `recommender.py`, where a number of songs are sampled from the test data in order to recommended the next song from the training data. The implementation simply takes a *track_id* value from the test data and predicts the cluster of the song accordingly. We then draw a song at random from the most popular songs belonging the same cluster. We carried out a very limited qualitative evaluation by examining and listening to 20 candidate songs for each recommendation. In this evaluation we found very inconsistent recommendations, with the 20 candidate songs spanning a very large variety of songs with seemingly no identifiable connection. The next logical step was therefore to reconsider the optimal clustering for the specific purpose of music recommendation. We hypothesized that due to the fact that each cluster simply spans to large of a variety of songs within the data set, it might be necessary to increase the number of clusters to reduce the intra-cluster distance, despite this giving sub-optimal silhouette score compared to the 6 clusters. Examining the original silhouette scores in the clustering range 2-32 clusters, we find that increasing the number of clusters only slightly worsens the silhouette score. With a configuration of 25 clusters we find a local maximum, with an average silhouette score of 0.134. We therefore re-evaluated the recommended songs with the new clustering, but still found the recommendation to be very inconsistent. For a more qualified evaluation of the recommendation, obviously a much larger scale qualitative experiment would have to be carried out. We therefore wihthold from making final conclusions about the recommendation, but at first evaluation it does not seem to perform very well. We also recommend that song recommendation based on other techniques such as a k Nearest Neighbor Search are explored for comparison.

V. Societal Impact and Conclusion

In this research project we set out to investigate if it is possible to produce models based on the supplied data set that are able to carry out two overall tasks : a) predict the attributes *danceability* and *genre/subgenre* of a song b) provide music recommendation based on a song or a listener mood. Our Random Forest Regressor model was somewhat able to accurately predict *danceability* and our Decision Tree model was quite successful in predicting *genre* and *subgenre* especially when trained on both audio-related features and transformed nominal attributes. Our model for clustering songs according to an overall mood was quantitatively evaluated to be rather good. Our clustering model for recommending songs based on a currently playing song performed rather poorly when evaluated qualitatively. One main reason that we were unable to provide more accurate recommendation, is simply the limited data Spotify makes available through their API, when compared to the data that Spotify has internally available. Our data set contains only few meta-attributes about the songs, but lacks data based on user behavior and listening patterns, which are extremely relevant for recommendation.

Without this data it is difficult to strike a balance between exploration and exploitation when recommending songs. Our clustering models do not seem to be able to capture what makes two songs sound alike (lacking exploitation). The genre prediction model on the other hand is not able to give recommendations outside the already categorised genres (lacking exploration).

We were intrigued to gain experience with building a recommender machine as it would make it possible to recommend music that can open new doors for listeners and gently push listeners out of their comfort zone. We experienced

some parts of the data set, e.g., the genre attributes, worked against this mission and might even re-produce stereotypical understandings of music and the culture it breeds and emerge from. This argument is based on a surface-level interpretation of the data set, and to determine if this is really the case further cultural studies must be conducted. Spotify's API does not detect more complex genres, such as Art Pop or Deconstructed Club Music, which is known for having an audience of historically under-served people. Thereby, when such genre categories cannot be comprehended in the data set, then the historically under-served groups of musicians and audience continue to be overlooked.

As a finishing note, we wish to acknowledge the climate impact training of machine learning models can have. Our data set is rather small, so the impact might be limited. However impact of training machine learning models should be kept in mind in times of climate crisis.

References

[1] J. Arvidsson, *30000 Spotify Songs: almost 30,000 spotify songs from the spotify api*, https://www.kaggle.com/datasets/joebeachcapital/30000-spotify-songs, Accessed: 2023-11-30.

[2] *Web API: spotify for developers*, https://developer.spotify.com/documentation/web-api, Accessed on: 2023-11-30.

[3] J. Han, M. Kamber, and J. Pei, "10 - cluster analysis: Basic concepts and methods," in *Data Mining (Third Edition)*, ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds., Third Edition, Boston: Morgan Kaufmann, 2012, pp. 443–495, ISBN: 978-0-12-381479-1. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00010-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123814791000101.