Make My Playlist

Seher Khan: 9273-0301-17
Sudeeksha Sanikommu: 3652-0422-72
Shreyasi Chaudhary: 2329-7387-80

ABSTRACT
The average American spends about 32 hours a week listening to music [0], however, it is natural to have less time available searching for new music. The aim of this project is to make it easier for a user to generate their playlist given a set of songs. We address this problem in two ways: first, we take a classification approach which creates the playlist based on predefined playlists; second, is a clustering approach which creates a more personalized playlist, taking into account the listening history of similar users. We conclude by performing a comparative study of the two approaches.

1. Introduction

Significance of music in our daily lives cannot be understated. The music industry is one of the largest industries in the world generating a revenue of $43 billion in 2017. Unsurprisingly, streaming applications generate the largest percentage of this revenue - Spotify alone had 83 million paying subscribers as of June 2018.
However, a countless number of new songs are released each day, making the task of listening to songs and selecting the best ones a tedious task. This tasks becomes more arduous when the playlist is tailored to a particular mood or purpose. Our aim is to make this task easier.
We begin our research by describing the data collection and cleaning. The next section on exploratory data analysis performed on our datasets.
Next, we apply two techniques. The **first approach** involves applying classification models to predefined playlists, given features of songs in those playlists. Given a seed song, it is then possible to classify the track into class(playlist) and suggest tracks for the user belonging that playlist class. The **second approach** employs clustering users based on their listening histories. Given a set of seed songs, it is possible to suggest the user tracks which have been listened to other users of his or her cluster.
We follow this by a discussion on previous work in this area, our conclusions and our individual contributions.

2. Approach 1 (Classification on playlists)

2.1 Data Collection

We started the process of dataset collection by looking at the Million Songs Dataset (MSD) [1]. The size of the entire dataset was 300GB so we decided to start exploring the data by taking a subset of the the MSD. The Labrosa website, which hosts the MSD, provides us a feature to download a randomly generated subset of the MSD consisting of 10,000 songs. The subset was available as a h5 file so we wrote python scripts to extract the audio features out of each of the songs in the subset and added it to comma separated file for easy manipulation.
We discovered that some of the important features like song popularity, danceability, etc. were missing in the subset so we decided to leverage the Spotify API [2] to collect such features from Spotify and then merge the two datasets. Spotify and MSD provided us many overlapping features but often had different values; in such cases we used the values given by Spotify data.
Next, we decided to download the other subsets from the Labrosa website and aggregate them all to construct our final dataset. The website stated the downloadable subset would be a random one. Hence, we assumed the songs in the new

dataset would not significantly overlap with the songs in the dataset we had previously collected. However, on downloading and processing the dataset we found it was the same one as before. It then became clear that only one subset is available with songs randomly chosen from MSD.

For the classification task, we required labels. We obtained labels in two ways.

Firstly, we collected data from a Columbia dataset [3] which lists songs, genres and some features. For additional features we pulled data from Spotify. Using the two resources we created a csv of of 48,086 data points, 48 attributes and 10 labels [**Dataset A**].

At the time of pulling data for Dataset A, we were unsure about the attributes in the dataset. On assessing this data, *we decided not to use Dataset A in our analysis*.

Secondly, we searched for 17 different playlists on Spotify and obtained lists of unique songs from one or more results. Each song was then assigned its respective playlist keyword as its label. We initially attempted to map them to the MSD dataset, to obtain the features of each song. However, we discovered that MSD did not have most the of songs released in the last 10-20 years. This led us to conclude that the MSD dataset is outdated for our application and hence discarded it for this approach. As a result, we extracted features for each song in our collected playlist from Spotify. We were able to create a final csv with 12,899 data points with 15 features and 17 labels [**Dataset B**]. This set was later cleaned and exploratory data analysis applied to it as explained in the following section and brought down to 8980 data points with 14 labels. *This dataset was used in our analysis for Approach 1.*

2.2. Data Cleaning and Exploratory Data Analysis

Making features of genre and artist categorical

The labels and artists had to be converted to categorical variables. This was easy to do for labels. In case of artists, it was more complicated.

For Dataset B, artists name were sourced from Spotify. Songs could have multiple artists, given in an array. We only extracted the artist listed first to limit the number of possible artists. After that, each unique artist was given a distinct number.

For Dataset A artists name were sourced from the Columbia dataset. Many song had multiple artists which were separated by various delimiters (comma, semicolon, slash, etc). To limit the number of distinct artists, we split the artist column by all delimiters and used only the first resulting column. Finally we assigned a distinct number to each artist name.

Eliminating Multi-labeling

In Dataset B, songs in playlists of one keyword overlapped with song in playlists of another. To avoid this, we removed all duplicate entry of songs in the dataset.

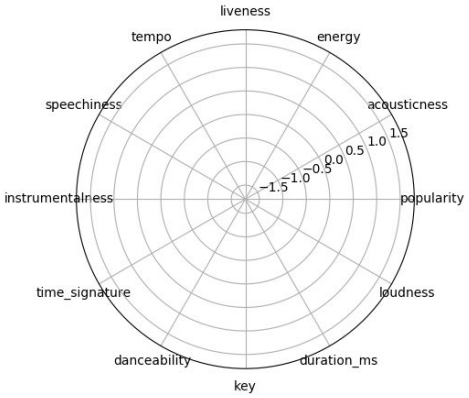
Descriptive Stats

We checked and did not find any missing values in Dataset A. We then proceeded to study descriptive statistics of the 14 numerical features of the dataset.

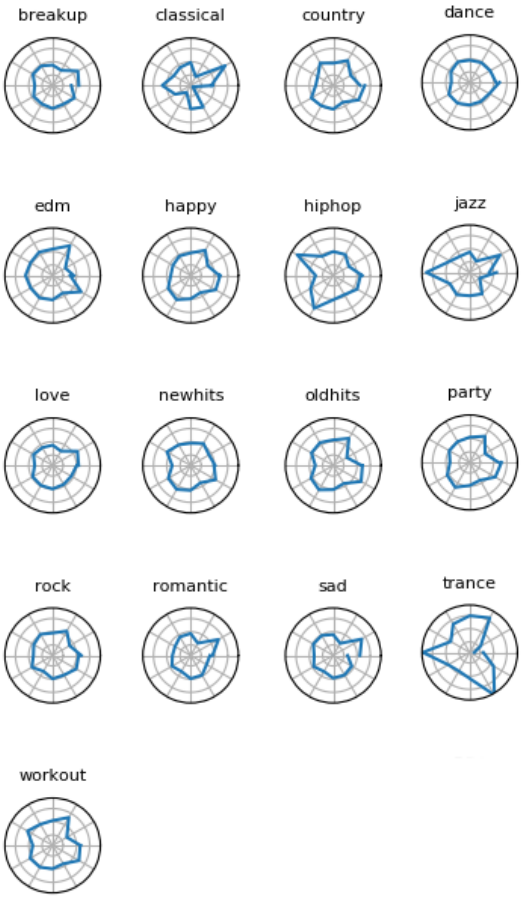
We found that the ranges of features varied significantly. For example, loudness varied from -42 to 1.5, popularity from 0 to 100 and energy from 0 to 1. For this reason, feature scaling was carried out. That is, the mean was pushed to 0 and standard deviation to 1.

To get a better feel of the data the scaled features were averaged and plotted on the following polar axis by label.

Make My Playlist



This showed the features illustrated expected distribution. For example, trance songs were long, had high instrumentalness and lower speechiness.



We also found statistical evidence at the 99% level to show that each feature was normally distributed. Naturally, this persisted after the features were scaled.

Correlation

Next, we calculated the correlation between each of our 15 attributes. We set the the absolute value 0.75 as the threshold indicating high correlation.

We did not find any columns with correlation greater than 0.75 on the positive or negative side. Thus, no columns were removed.

Merging similar labels

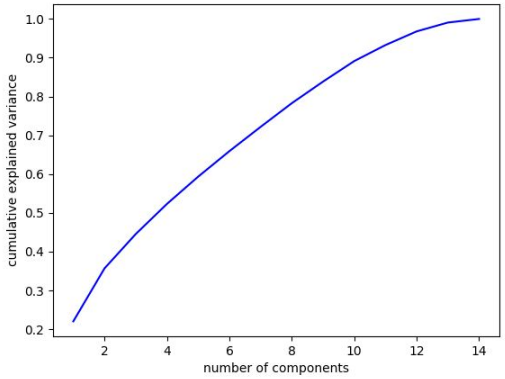
While we were removing duplicate songs during data cleaning to avoid multi-labelling, we noticed certain keywords had a large amount of overlap. Such as love/romantic and breakup/sad. As a result we merged them under “romantic” and “sad”, respectively.

Dimensionality Reduction through PCA

Whenever possible, dimensions should be reduced to avoid overfitting. We therefore attempted PCA on our scaled Dataset A. We used two criteria to reduce dimensions:

- Consider components upto the point where cumulative explained variance is greater than 90% => selected the first 11 components as these explain 96% (see graph below)
- Consider all components with explained variance greater than 1 => selected first 4 components[1]

[1] [3.07694483, 1.71038503, 1.23143809 1.06806714, 0.98117303, 0.90750388, 0.87586567, 0.85473453, 0.76605287, 0.58498558, 0.49457655, 0.31962493, 0.13009569]



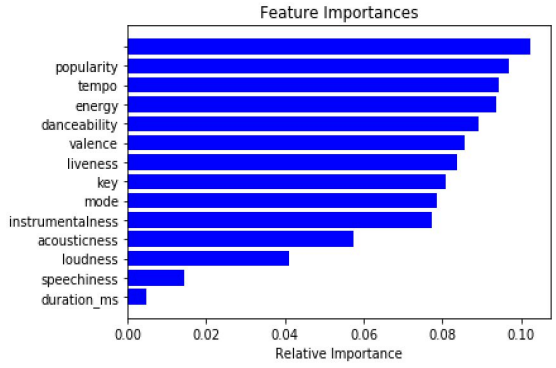
For data visualization, it is common to project the data on the first two components and create a scatter plot. In our case this was not useful as the first two components only explained 36% of the variance.

Note that for data modelling, the raw data is used rather than the transformed data produced by PCA, unless specified, otherwise.

Feature Importance with Random Forest

Another attempt to reduce dimensionality was made by applying Random Forest to calculate the importance of each feature in our dataset. In this approach, multiple decision trees are built and the importance for each of the nodes is found by calculating the impurity reduction by that node across all the generated trees. The impurity reduction is computed as the sum over the number of splits that include the feature, proportionally to the number of samples it splits.

The following results were obtained on applying the algorithm.



From these results, it can be observed that duration, speechiness and loudness features have significantly low importance compared to other features. We apply classification algorithms on our dataset by removing "unimportant" features and observe the prediction accuracy.

2.3 Methodology

2.3.1 Classification Algorithms

We took the **Dataset B** (consisting of tracks from all the different playlists and their audio features) and processed it as explained in 3.1.

Initially we worked with 10 fold cross validation for calculating accuracies. However, to make the most of our small dataset, we have instead used 5x2 cross validation.

We also employed the transformed data from PCA for modelling (Naive Bayes, LDA and QDA). However, due to little/no improvement in accuracies, we did not attempt to use PCA in all models.

2.3.1.1 Naive Bayes

We applied a Gaussian Naive Bayes model to our data considering all features were found to be normally distributed. Since this model does not have any hyperparameters, no tuning was required.

Average accuracy over 10 folds was found to be low at 38.3%. We then used the transformed output of PCA and applied NB with cross validation retaining all, 10 and 4 components. The average accuracies were 40.0%, 34.6%, and 37.6% respectively.

Since there was no significant increase in accuracy from PCA, we dropped its use for this model.

We also applied Gaussian Naive Bayes with 5x2 CV. This resulted in an average accuracy of 38.6%.

2..1.2 Decision Tree

This algorithm constructs a tree consisting of root node, child node and leaf node by employing the concept of entropy for selection of nodes. We employ various techniques for tuning our classification model by running various tests on the validation dataset. The first parameter we tried varying was the maximum possible depth of the decision tree. Deeper trees capture more information about the model but increase the complexity of the model and thus increase chances of overfitting. We ran our algorithm by varying the max depth from 1-50. We got the best validation accuracy of 34.8% at depth = 13, which aligns with the fact if the tree is too deep, overfitting increases.

The next parameter we varied was the minimum number of samples required to split the node. A very low value of minimum number of samples required at each node can lead to overfitting and a very high number will cause underfitting as the model is unable to learn. We vary this parameter from 10% of the samples to 100% of the sample and check it performance on the validation set. The best of obtained validation accuracy was 30% at the value 10% of the sample. The last parameter we varied was the maximum number of features to consider for each split. We varied this from 2 to total number of attributes in our dataset and got the best validation accuracy of 35.49% when we use a maximum of 7 features for split. Using all these parameters in conjunction gave a us a model with validation accuracy of 35.8%. To improve the accuracy, we used grid search to get the best combination of hyperparameters and using 5x2 cross we received a validation accuracy of 39.08%

2.3.1.3 Support Vector Machine

This algorithm plots each of the data items on an n-dimensional plane where each dimension represents a feature; and, outputs an optimal hyperplane that separates the classes.

Initially, the algorithm was applied on our dataset to get an accuracy of 40.5%. An average accuracy for Stratified 10 fold cross validation was found to be 45.5%.

In order to find the parameters for the model that gave the best accuracy, Grid search was applied. Grid search is a method for hyperparameter optimization that performs an exhaustive search through all the specified hyperparameters to find those giving the best results. On applying Grid search, it was found that the best model giving highest accuracy of 43%, was a model with C parameter set to 1 and kernel set to ‘rbf’. In general, RBF or Gaussian kernel is said to perform better than linear kernel; which is also observed in this case.

On applying 5 times 2 fold cross validation on the best obtained model from grid search, an average accuracy of 47% was obtained with average precision, recall and f1-score of 0.47, 0.47 and 0.47 respectively. It is observed that applying Grid Search helped in finding the best model that gave the best results

2.3.1.4 Logistic Regression

This algorithm uses a logit function that outputs the probability prediction; i.e, along with prediction the class label, it also outputs the probability of the class.

On applying the algorithm with default parameters on our dataset resulted in an accuracy of 37.5%. On explicitly specifying parameter, multiclass as multinomial, the accuracy increased to 40.4% with average precision=0.42, average recall= 0.46 and average f-1 score= 0.34. On applying Stratified 10 fold cross validation, average accuracy of 43.5% was obtained.

In order to find the parameters for the model that gave the best accuracy, Grid search was applied. On performing Grid search, it was found that the

best accuracy of 41% was achieved for C=10. On applying 5 times 2 fold cross validation on the best obtained model from grid search, an average accuracy of 45.4% was obtained with average precision, recall and f1-score of 0.45, 0.45 and 0.45 respectively. It is observed that applying Grid Search helped in finding the best model that gave the best results.

2.3.1.5 Linear Discriminant Analysis

Logistic regression is the preferred classification technique for a two class problem. However, with increasing classes, a LDA or QDA model is generally more suitable. Since this model does not have any hyperparameters, no tuning was required.

Applying the model 10-Fold cross validation resulted in an accuracy of 40.5%. Using transformed output of PCA with all, 10 and 4 components resulted in an accuracy of 40.5%, 32.8% and 37.4%, respectively. Since there was no significant increase in accuracy from PCA, we dropped its use for this model.

We also applied the model with 5x2 CV. This resulted in an average accuracy of 41.9%.

2.3.1.6 Quadratic Discriminant Analysis

Since this model does not have any hyperparameters, no tuning was required. Applying the model with with 10-Fold cross validation resulted in an accuracy of 39.4%. Using transformed output of PCA with all, 10 and 4 components resulted in an accuracy of 39.4%, 34.2% and 38.0%, respectively. Since there was no significant increase in accuracy from PCA, we dropped its use for this model.

We also applied the model with 5x2 CV. This resulted in an average accuracy of 40.6%.

2.3.1.7 Random Forest

Random forest classifier creates a set of decision trees from randomly selected subset of the training set. It then takes a majority votes among all decision trees to decide the final class of the given set of attributes. Like decision tree we varied the parameter, maximum possible depth of each individual tree. In general, it was observed that increasing the maximum depth of the tree, increased the accuracy upto a point after which it started reducing. Using Grid search we discovered that the best model is has a maximum depth of 11, minimum samples required for split equal to 3 and the number of estimators set to 200. The average accuracy obtained on performing 5x2 CV using this model was 47.68% with an F1 score of 0.44.

2.3.1.8 Voting Classifier

Voting Classifier is a type of ensemble classifier that combines various machine learning algorithms, takes all the labels predicted by the algorithms and outputs the label that has most votes (i.e, majority label). This type of voting classifier is termed as a ‘Hard’ voting classifier. On applying this classifier in combination of Logistic Regression, Gaussian Naive Bayes, SVM and Random Forest classifiers, with 5 times 2 fold cross validation, we obtained an average accuracy of 47.3% and average precision, recall and f1-score of 0.47, 0.47 and 0.47 respectively. These results are compared with the average accuracies obtained with 5 times 2 fold cross validation for SVM(47%), Logistic regression(45.4%), Random Forest(47.6%), Decision Tree(35.5%) and Gaussian Naive Bayes(38.6%). It is observed that Voting Classifier gives the best possible accuracy of all the algorithms individually considered.

2.3.1.9 Neural Networks

Neural Networks have gained popularity in the past few decades. It functions similar to that of a human brain, consisting of a large network of interconnected nodes. A basic neural network

consists of an input layer, one or more hidden layers and an output layer.

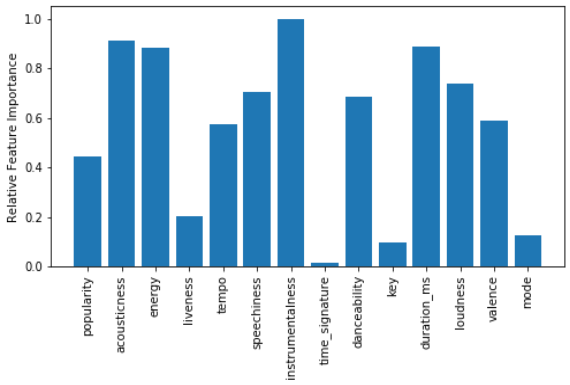
We have decided to go with a keras neural network. Since our dataset has multiple classes, 'one hot encoding' was applied, which converts the output values into categorical values that represent the numerical value of the outputs. Then, a neural network was defined with a sequential model having one input layer, one output layer and two hidden layers. A Keras Classifier is used to build the model with epoch value set to 50 as the loss remained close to same after that. On performing 10 fold cross validation for this model, an accuracy of 47.42% was achieved with standard deviation of ± 0.4% .

2.3.1.10 Gradient Boosting Classifier

This technique fits several weak learners (trees) to the data and then consolidates their results. It consists of several hyperparameters that must be tuned to produce a useful prediction.

With default parameters, gradient boosting classifier resulted in an accuracy of 48.7%.

Grid search was then used to tune parameters. It was found that optimal accuracy was obtained where the number of learners was 100 (n_estimators) with each learner having a maximum depth of 7 (max_depth), minimum number of examples to be assessed for splitting node were 350 (min_samples_split) and minimum examples to be present in a leaf node were 100 (min_samples_leaf). Using these parameters, 5x2 CV produced an accuracy of 50.7%. We finally applied this model to the entire training set and then made a prediction on the test set. The accuracy we achieved was 51.2%. The following figure shows the relative feature importance estimated by the last model.



2.3.1.11 Rest Vs One Classification

The models previously considered were all applied on a subset of Dataset B where all duplicate songs had been removed. That is, it was naively assumed that a song can only belong to a single playlist. In actuality, a song can belong to several playlists. That is the problem is a multilabel classification problem. The Rest Vs One Classification technique is well suited to such a problem. This fits a binary classifier to each class and the accuracy for this approach is the subset accuracy. The subset accuracy means that for a prediction to be true, each label of the sample must be correctly predicted.

We took our complete Dataset B and applied this technique with 5x2 cross validation. For the binary classification of each playlist, we employed some models; namely Naive Bayes (accuracy: 37.8%), LDA (accuracy: 40.3%), QDA (accuracy: 38.9%), Logistic Regression (accuracy: 42.8%) and Gradient Boosting Classification (accuracy with default parameters: 48.5%).

2.3.1.12 AdaBoost Classifier

The aim of the AdaBoost Classifier is to convert a set of weak classifiers into a strong classifier. Adaboost iteratively retrains the algorithm by choosing the training set based on the accuracy obtained in the previous iteration. It assigns weight to each trained classifier based on the accuracy achieved. The adaboost algorithm was

tried with the following algorithms and the results obtained our summarized below. The results obtained are averaged out values from 5 runs of 2 fold cross validation.

Algorithm	Accuracy
Decision Trees	46.9%
Random Forest	45.13%
SVM	13.7%

The results show that Adaboost works well using weak learners like decision trees, but does not give good results when used with strong learner like SVM.

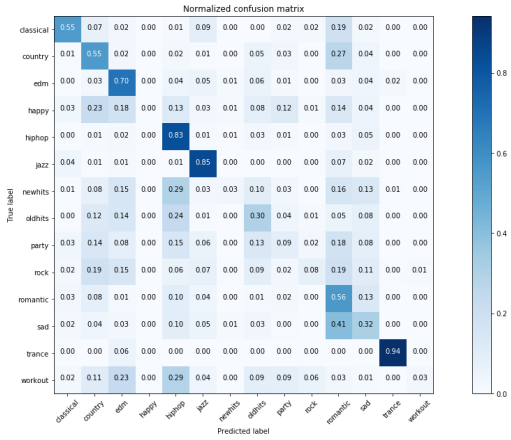
2.3.1.13 Using class Hierarchies

This approach is based on the paper [7]. The authors in this paper describe an an alternative way to address the multiclass problem by hierarchically distributing the classes in a collection of multiclass subproblems by reducing the number of classes involved in each local subproblem. The steps we have used are as follows:

1. Trained a random forest classifier as multiclass problem with 14 classes. The accuracy of this classifier obtained via 5 runs of 2 fold cross validation was 46.5%.
2. Generated a confusion matrix by running the classifier mentioned in 1. The confusion matrix obtained is displayed below.
3. The confusion matrix obtained in 2 was then normalised. For a confusion matrix M, the normalization was performed using the equation below:

$$\overline{M}_{ij} = \frac{M_{ij}}{\sum_{j=1}^{n} M_{ij}}$$

The matrix obtained as a result of normalization was titled normalised confusion matrix.



4. The degree of overlap between classes i and j in the normalised confusion matrix was calculated using the equation below:

$$overlap(i,j) = \begin{cases} \frac{\overline{M}_{ij}+\overline{M}_{ji}}{2} & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

As a result of using the above equation on the normalised confusion matrix, we obtained the overlapping matrix.

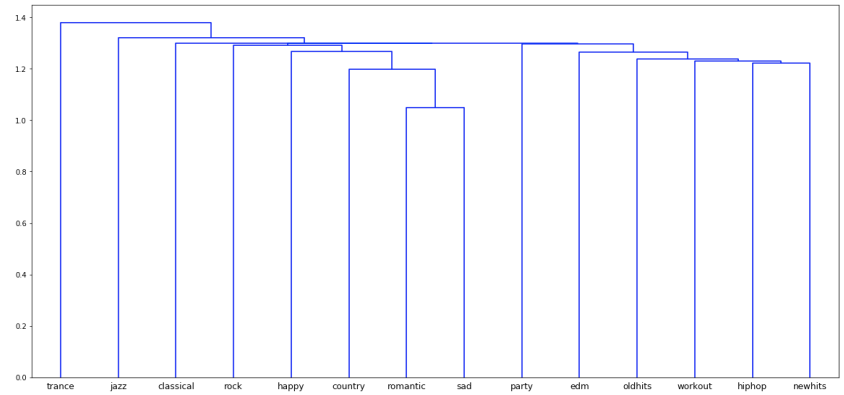
5. Next we calculated the similarity between the classes i and j using the following equation:

$$d_S(i,j) = 1 - overlap(i,j)$$

The values of dS(i, j) fall in the range of [0, 1]. Values close to 0 indicate a very high level of similarity between the classes and values close to 1 indicate a low level of similarity between the classes. The matrix obtained as result is referred to as the similarity matrix.

6. Since, our goal is to create a hierarchy between the classes, we send the similarity matrix as input to the agglomerative clustering algorithm. The 14 classes are clustered using

agglomerative clustering on the basis of the average linkage(distance) between the similarity values of the classes. The dendrogram obtained using average linkage between the similarity values of the classes is displayed below.



7. Using the dendrogram above, a pipeline of classifiers was created as shown below.

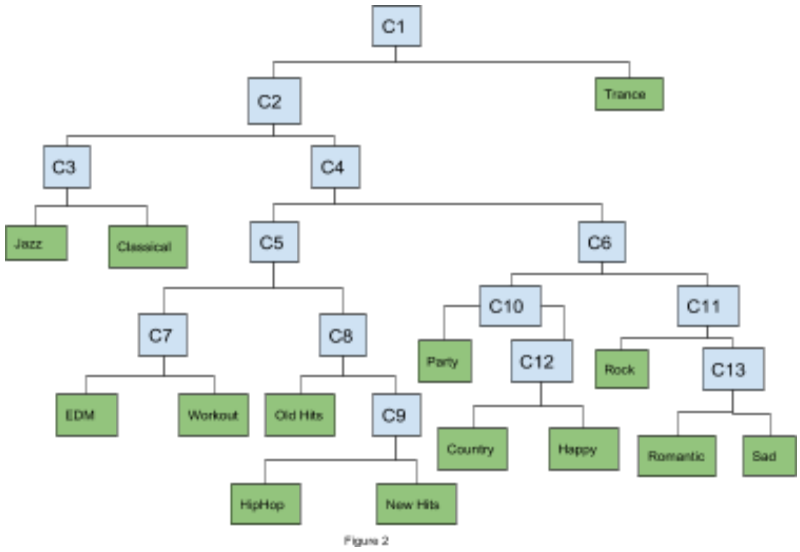
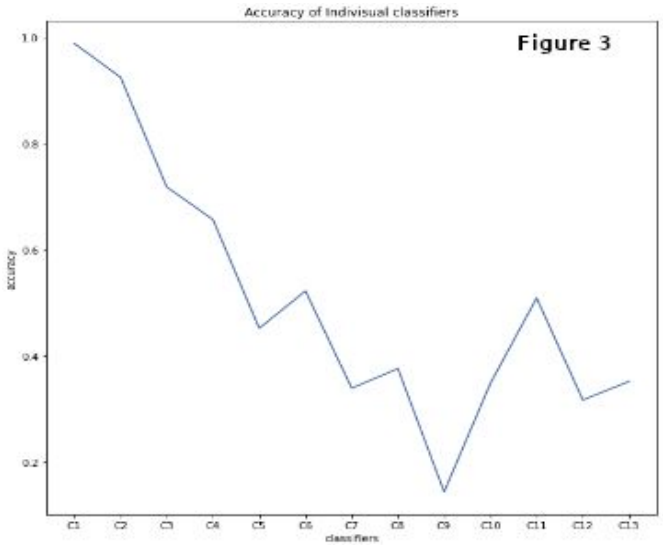


Figure 2

Each of the classifiers shown in figure 2 were trained individually and their respective accuracies based on 5 fold cross validation were as follows:

classifier	accuracy	Classifier	accuracy
C1	98.73%	C8	80%
C2	91.75%	C9	71%
C3	87.2%	C10	66%
C4	74%	C11	85%
C5	78%	C12	77%
C6	70%	C13	65%
C7	74%		

Next, we performed 5 runs of 2 CV through the pipeline of classifiers shown in figure 3. We discovered that when the same classifiers are run through a pipeline, due to errors in the previous stages the accuracy of each of the classifiers used at later stages of the pipeline drops significantly. Figure 3 describes the accuracy of the individual classifiers when used in a pipelined manner.



All of the classifiers are trained using gradient boosting algorithm. The final accuracy obtained through 5x2 cross validation is 39.7% which is much lower than most of our models.

2.4 Performance Metrics

We are considering the following parameters for evaluating our models: accuracy, macro, micro and weighted average of precision, recall and F1 score.

The accuracy measure tells us the percentage of points classified correctly. The macro value of precision, recall and F1 score gives us the average of the 3 values for each class. Micro average will aggregate the contributions of all classes to compute the average metric. We also use weighted macro average in which is each class's contribution to the average is weighted by the relative number of examples available for it. Since, we are dealing with unbalanced classes we give more importance to micro and weighted aggregates as they provide us a better way to analyse our results. We have decided to use decision trees as baseline for evaluating our future models. The results summarized below are based on 5x2 cross validation.

Validation Performance

Algorithm	**Accuracy**	**Micro F1**	**Micro precision**	**Micro recall**
Decision Tree	39.08%	0.38	0.38	0.38
SVM	47%	0.47	0.47	0.47
Logistic Regression	45.4%	0.45	0.45	0.45
Naive Bayes	38.6%	0.39	0.39	0.39
LDA	41.9%	0.42	0.42	0.42
QDA	40.6%	0.41	0.41	0.41
Random Forest	47.68%	0.48	0.48	0.48
Gradient Boosting Classifier	**50.7%**	**0.51**	**0.51**	**0.51**
One vs Rest Classification	48.5%	0.49	0.49	0.49
Neural Networks[2]	47.42%	-	-	-
Ada boost	46.9%	0.47	0.47	0.47
Voting Classifier	47.3%	0.47	0.47	0.47
Class Hierarchy	39.7%	0.39	0.39	0.39

2.5 Test Performance

Since the size of our dataset is small(8990 rows), we decided to shuffle our dataset, train our best performing model, the gradient boosting classifier on randomly selected 90% of data points and then test our algorithm on the 10% data points.

Metric	**Value**
Accuracy	51.22%
Micro Precision	0.5122
Micro Recall	0.5122
Micro F1 score	0.5122

The Precision, recall and F1 score the the average values of precision, recall and F1 score of each of the classes. Note that all scoring parameters are equal, this occurs because we use "micro-averages" in calculating these measures, as pointed out by the sklearn documentation. We used micro averages because they were more

```
print 'Gradient Boosting: performance on test data of model fitted to training data in one go'

min_samples_split = 350
min_samples_leaf = 100
max_depth = 7
max_features = 'sqrt'
subsample = 0.8

gb1=GradientBoostingClassifier(min_samples_split=min_samples_split,min_samples_leaf=min_samples_leaf,
                               max_depth = max_depth, max_features =max_features,subsample=subsample)
gb1.fit(X_trainset,y_trainset)
pred = gb1.predict(X_testset)
score = precision_recall_fscore_support(y_testset, pred, labels=range(14), average='micro')
print 'accuracy =', accuracy_score(y_testset, pred, normalize=True)
print 'prediction =', score[0]
print 'recall =', score[1]
print 'f_score =', score[2]
```

```
Gradient Boosting: performance on test data of model fitted to training data
accuracy = 0.512249443207127
prediction = 0.512249443207127
recall = 0.512249443207127
f_score = 0.512249443207127
```

[2]Implementation of f1-score, precision and recall is not available for neural networks with keras classifier on sklearn

appropriate to our multi-class problem with imbalanced classes.

3. Approach 2 (Using listener's history)

3.1 Data cleaning

For Approach 2, we required the listening history of users. This dataset was obtained from the Echo Nest Taste Profile dataset [5]. This dataset consists of approximately 40 million data points with the attributes: SongId, UserId and Play Count. We matched the first 4 million of these data points with our previously collected MSD dataset with 48,086 songs and 39 attributes. The number of unique users in our dataset was 54,697. For each user, we took a weighted average of all the songs the user has played. The weights were number of times the user has played the songs. This gave the attributes of the song that the user has played more frequently, proportionally higher weightage. We finally obtain 54,697 data points and 23 attributes, each which are representative of a particular user's listening history [**Dataset C**].

3.2 Exploratory Data Analysis

We begin our Exploratory data analysis by checking for null or missing values. Fortunately, we did not find any. Next, we calculate correlation between each of our 39 attributes. We set the the absolute value 0.75 as the threshold indicating high correlation.

If we find a pair of columns with correlation greater than +0.75 or less than -0.75, we drop one of those of columns.

Based on the correlations we removed the following attributes:

- tempo
- Time_sig
- Danceablity
- Avg_timbre 1
- Duration_ms
- Valence
- Var timber 2, var timber 4-12

In total, we removed 15 attributes and are left with 23 attributes.

Assessing descriptive stats did not show any extreme values for our features. This was not surprising since the quality of Spotify dataset is high. If any outliers exist, they are all valid data points. As a standard machine learning practice, and because popularity ranged from 0 to 100 (while other variables are much smaller), we standardized our data by bringing the mean of each feature to 0 and standard deviation to 1.

3.3 Methodology

We calculated a representation of each user's listening history by taking weighted average of the song features based on the number of times the user has listened to song. Our final goal is to cluster the users based on their listening history and recommend the songs listened to by other users in the same cluster.

We followed the following steps:

1. Calculates weighted average of song features of all the songs listened to by a user. We assigned higher weights to songs played more number of times. As a results, we obtained a row for each user which is representative of his or her listening history.
2. We separated our dataset(users) into training, test and validation in a 70-20-10 ratio.
3. We clusterted the users in the training set using the following algorithms.
    - K Means Clustering
    - Mean Shift Clustering
    - Affinity Propagation

4. For each user in our validation set, we used 60% of songs as seed songs and kept the rest 40% to check how many songs have been predicted correctly. This set was referred to as the checking set.
5. We used the clustering model, to assign each user in the validation set a cluster.
6. For each user in the validation set, we took all the songs listened to by other users belonging to that cluster that were not present in the user's seed set, and presented it as the songs that belong to this user's playlist(listening history)
7. The last step was to calculate the accuracy by finding out the number of songs in the user's checking set that are also present in the result of 6.

We tried techniques like PCA for dimensionality reduction but this did not help us improve our accuracy. We further tried to optimize each of the clustering algorithms to the best our ability, yet none of the songs in the validation checking set matched the songs predicted by our algorithm. We believe that this could be due the following reasons:

1. Number of songs listened to by a user can be as low as 5.
2. Large number of users present in our training set so the clusters created might not have been able to represent the each user's listening history well.
3. Weighted average based solely upon the listening history might not have been the best way to represent the user's listening history.

4. Related Work

A plethora of literature exists for the problem of playlist generation and genre classification. The problem has either been approached as a classification problem or a graph problem.

In case of classification, either binary (song belongs to playlist / does not belong to playlist) or multiclass-classification (each playlist is a seperate class) is implemented.

Multiple graph approaches have also been used. Firstly, each song is considered a node and clustered based on its features, playlists are recommended by predicting the nearest neighbours to the song in the cluster [4]. Secondly, along with considering songs nodes, the order in which they have been played is also incorporated by making a directed graph and setting edge weights as transition probabilities [5]. The recommendations are given by:

i) finding the shortest path from the seed song to the next "n" songs, or

ii) given the first and last song, finding the shortest path between them.

The features in this models are usually audio features of the song. Along with using features directly from datasets like MSD, papers attempt to extract their own features, eg. using Latent Dirichlet Allocation and Hidden Markov modeling on timbre segments, and by performing sentiment analysis on lyrics [4].

Finally, playlists histories were also considered. For example, if an artist appeared in a playlist, all songs of that playlists were associated with songs of that artist [6].

5. Conclusions

We have successfully come up with an approach that can automatically create a playlist given a set of seed songs to begin with. Our best performance was obtained using the gradient boosting algorithm. We verified our results by running our model through 5 rounds of 2 fold cross validation achieving an accuracy of 50.7% ± 0.2% and F1 score of 0.51 ± 0.002. Our test accuracy was 51.22% and F1 score was 0.5122. We are optimistic about this result as the classification is predicted on a total of 14 classes. Furthermore, we explored various

different approaches to improve the accuracy of multiclass classification with a large number of classes.

6. Contributions

Seher:

- Modeled data using Naive Bayes, LDA and QDA, Gradient Boosting, One Vs Rest classification for approach 1 and K-Means clustering for approach 2 (also attempted clustering for approach 1).
- Utility Scripts for approach 2
- Conducted literature review
- Studied the MSD dataset, extracted and uploaded to Firebase via Python to allow a clearer view of the features for selection.
- Wrote and ran multiple scripts to fetch data from Spotify and combine it with MSD, MSD genre and playlists datasets
- Data cleaning, formatting and exploratory analysis for Approach 1 (including PCA)

Sudeeksha:

- Modelled data using SVM, Logistic regression, Voting Classifier, Neural Networks for Approach 1
- Affinity propagation for Approach 2
- Wrote and ran script to fetch data from Spotify for songs and playlists from Spotify for creating dataset for approach 1
- Worked on generating correlation matrix and finding correlations
- Worked on computing feature importance using Random Forest for Approach 1 and identified “important” features
- Found datasets to use, particularly the MSD genre dataset
- Wrote and ran script to fetch data of lyrics for songs
- Wrote scripts to replace user song id to match track id in MSD
- Led presentation making activity
- Utility Scripts for approach 2

Shreyasi:

- Created the class hierarchy and pipeline of classified for section 4.2.1.12.
- Modelled data using Decision Tree, random forest, adaboost and class hierarchy.
- Mean Shift Clustering for approach 2
- Identified datasets to be used
- Wrote script to extract features from MSD and store them in a CSV file
- Combine the features from the CSV obtained from MSD with features from spotify.
- Performed majority of EDA for Approach 2
- Wrote a script that removes all the songs that the user has listened to which are not present in the MSD dataset from the user listening history dataset.
- Wrote scripts for creating each user’s playlist based on their listening history for approach 2
- Wrote scripts to calculate the weighted average of the songs features in the user playlists.
- Led report writing activity
- Utility Scripts for approach 2

7. Bibliography

[0] Time with tunes: how technology is driving music consumption [Web log post]. Retrieved September 11, 2018, from https://www.nielsen.com/us/en/insights/news/2017/time-with-tunes-how-technology-is-driving-music-consumption.html

[1]Million song dataset. https://labrosa.ee.columbia.edu/millionsong

[2] Spotify Web API. https://developer.spotify.com/documentation/web-api

[3] MSD genre dataset. https://labrosa.ee.columbia.edu/millionsong/blog/11-2-28-deriving-genre-dataset

[4] Keith, K., & Fassois, D. (2005) Automated Playlist Generation. https://pdfs.semanticscholar.org/759a/656385d84b160574a3c0231294d925026490.pdf

[5] Logan, B. (2002) Content-Based Playlist Generation: Exploratory Experiments. https://www.researchgate.net/profile/Beth_Logan/publication/220722880_Content-Based_Playlist_Generation_Exploratory_Experiments/links/5429f1bf0cf27e39fa8e6c90/Content-Based-Playlist-Generation-Exploratory-Experiments.pdf

[6] Bonnin, G., Jannach, D. (2015) Automated generation of music playlists: Survey and experiments. https://www.researchgate.net/profile/Geoffray_Bonnin/publication/271727945_Automated_Generation_of_Music_Playlists_Survey_and_Experiments/links/57ea820208aeafc4e88a4930/Automated-Generation-of-Music-Playlists-Survey-and-Experiments.pdf

[7] Improving Performance of Multiclass Classification by Inducing Class Hierarchies https://www.sciencedirect.com/science/article/pii/S1877050917308244